Recitation_08-Logistic_Regression-2022

March 29, 2022

1 Recitation 8: Logistic Regression

This recitation is due on Friday April 1

1.1 Case Background

In this lab, we continue the analysis of the metal casting problem from the previous recitation. Recall that, as part of a longer manufacturing process, a factory needs to cast a large number of rectangular metallic blocks. The blocks are manufactured using a mold, consisting of the main cavity, a cup through which the molten metal is poured, and two risers for cooling. The size and shape of the pouring cup and risers affect how quickly the metal can be poured into the mold, how quickly it cools, and whether it sets correctly.

1.2 Objective

The factory needs to cast batches of 100 blocks of size 4.5×4.5×7 inches. The current casting approach is conservative – it takes a long time to pour, but the blocks always set correctly and are usable. Your goal is to achieve a significant reduction in average casting time while still ensuring that most blocks are usable.

1.3 Variables describing the mold

The following nine variables describe the shape of the mold: * Riser Height * Riser Diameter * Riser 1 Position * Riser 2 Position * Gate Diameter * Cup Height * Sprue Height * Sprue Diameter Bottom * Sprue Diameter Top Together we call these the "mold variables".

We will vary these parameters to choose a new shape for the mold to reduce average casting time while ensuring most blocks are usable.

1.4 Data

To obtain data on how the mold variables affect pouring and cooling, a batch of 100 blocks are poured while randomly varying the nine mold variables around their baseline values. The data is available in castdata.csv. This is the same file as the previous recitation. As before, the first nine columns are the mold-variables listed above. The 10th column is `BatchTime` (how long the molten metal took to cool inside the mold). The 11th column is `Feasible`. `Feasible` is 1 if the block that came out of the mold was usable and 0 if it wasn't.

1.5 ASSIGNMENT

This lab project will guide you through the analysis. First, we must import castdata.csv into the notebook.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split, cross_val_score
df = pd.read_csv('castdata.csv')
df.head() # view the data
```

[1]:

	GateDiam	CupHeight	SprueHeight	RiserDiam	SprueDiamBot	SprueDiamTop	\
0	0.500000	5.000000	10.000000	7.000000	0.500000	0.750000	
1	0.318317	5.604768	10.148839	6.745247	0.635632	0.889822	
2	0.353834	5.846289	9.528183	7.345273	0.642489	1.005683	
3	0.596163	4.280124	10.887751	6.952856	0.317180	0.589558	
4	0.484338	5.205026	11.003688	7.254450	0.531424	0.595769	

	RiserHeight	Riser1Pos	Riser2Pos	BatchTime	Feasible
0	5.000000	3.500000	5.000000	3031.292318	1
1	5.318354	3.168774	5.233322	3042.417820	0
2	4.836570	3.265961	5.482437	3050.300487	1
3	5.492964	4.029533	4.719791	3034.202547	1
4	4.897376	4.011947	4.822747	3033.046509	1

Now we compare the distributions of the predictors (the mold variables) among two subsets of the data: those with Feasible = 0 and those with Feasible = 1.

Before doing this, we first normalize each of the predictors by subtracting off its mean value and dividing by its standard deviation. This puts all of the predictors on the same scale, so that we can plot them together. (The boxplot() function we use below doesn't allow multiple y axes, so if we didn't do this then we would mostly see that the different predictors have different mean values, which wouldn't be very informative.) Also, by subtracting off the mean value and dividing by their standard deviation, we can more easily see how changes in these values, within the range evaluated in our data, affect feasibility.

[2]:
```
normalized = df.copy()
predictors = np.delete(normalized.columns.values,[9,10]) # All columns except␣
 ↪Feasible and BatchTime are predictors
for i in predictors:
    # Normalize all predictors
    normalized[i]=(normalized[i]-normalized[i].mean())/normalized[i].std()

normalized.head()
```

[2]:

	GateDiam	CupHeight	SprueHeight	RiserDiam	SprueDiamBot	SprueDiamTop	\
0	0.093566	-0.038859	-0.112868	0.037250	-0.051764	-0.078771	

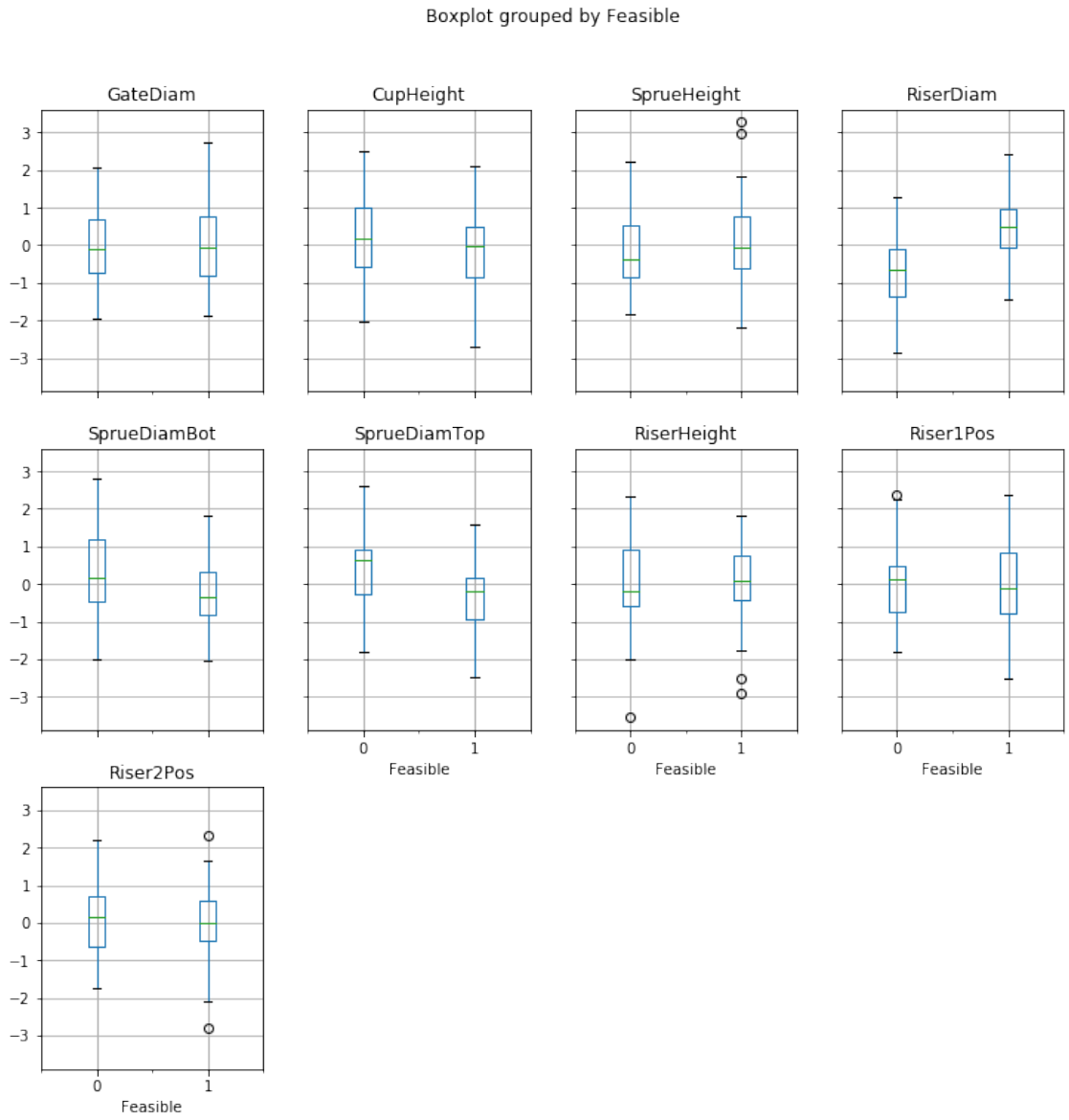
```
1 -1.161374   1.499244     0.155546  -0.428747      1.055643      0.698712
2 -0.916050   2.113506    -0.963736   0.668825      1.111626      1.342958
3  0.757796  -1.869717     1.488091  -0.048987     -1.544448     -0.970910
4 -0.014619   0.482582     1.697170   0.502691      0.204806     -0.936376

   RiserHeight  Riser1Pos  Riser2Pos    BatchTime  Feasible
0     0.091616   0.024610   0.149696  3031.292318         1
1     0.818235  -0.855144   0.653394  3042.417820         0
2    -0.281401  -0.597010   1.191190  3050.300487         1
3     1.216770   1.431078  -0.455224  3034.202547         1
4    -0.142616   1.384370  -0.232962  3033.046509         1
```

We now make a boxplot showing the distribution of each predictor in the two subsets of the data (those with Feasible=0 and Feasible=1).

If you haven't used boxplots before, here is how they work. A boxplot uses a box to visualize a collection of observations. In the center of the box is a green line. The height of this line is the median of the observations. The height of the top of the box is the 75% quantile (the number such that 75% of the observations are below it). The height of the bottom of the box is the 25% quantile. There are lines coming out of the box (called "whiskers") that represent the range of values over which most of the data falls. There may be a few circles above and below the whiskers, which are data that fall outside of this range. These points are called "outliers".

[3]:
```
normalized.boxplot(column = predictors.tolist(), by = 'Feasible', layout=(3,4),␣
↪figsize=(12,12))
# layout=(2,2) configures how the plots will be outputted into the console
# for more info on boxplot() head to the link:
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.
↪boxplot.html
```

[3]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fedc1003150>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fed61da7c10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fedc1023bd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fed80944510>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7feda060a990>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fed61dcfcd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fedc1098dd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fedc10cf610>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fed61e14190>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fed61e47790>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7feda0664750>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7feda0697f50>]],
      dtype=object)
```



Based on the boxplots above, we can get a rough sense for which variables have the biggest effect on feasibility. To do this, we can look at the boxes for a predictor to see how the quantiles (25%, 50%, 75%) differ between the two subsets: Feasible = 1 and Feasible = 0. If each of these quantiles is similar, so that the two boxes have the same top, green line, and bottom, then the predictor probably doesn't have much effect on Feasible. But if some or all of these aspects of the box are very different from each other, then the predictor likely has a bigger effect.

1.5.1 Question 1

Based on the boxplots above, which two or three variables have the biggest difference on Feasible? For those variables, are they higher or lower on average when Feasible = 1 rather than 0? Use these observations to make an initial recommendation about values of the mold-variables that will

maximize the probability of a feasible casting.

1.6 Logistic Regression

We will fit a logistic regression model to estimate the probability that a casting is feasible using the mold-variables as predictors. Fit an initial logistic regression model, using the following commands:

[4]:
```
# pull out our response variable
y = df["Feasible"]
# df[predictors] is a dataframe that consists only of the predictors
# sm.add_constant adds a column of 1's and allows sm.Logit to fit an intercept
X = sm.add_constant(df[predictors])
# fit the model and return the regression results
model = sm.Logit(y, X).fit()
results = model.summary()
results
```

```
Optimization terminated successfully.
         Current function value: 0.293880
         Iterations 8
```

```
/usr/local/anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:2495:
FutureWarning: Method .ptp is deprecated and will be removed in a future
version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

[4]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results
==============================================================================
Dep. Variable:               Feasible   No. Observations:                  100
Model:                          Logit   Df Residuals:                       90
Method:                           MLE   Df Model:                            9
Date:                Sun, 27 Mar 2022   Pseudo R-squ.:                  0.5699
Time:                        21:04:01   Log-Likelihood:                -29.388
converged:                       True   LL-Null:                       -68.331
Covariance Type:            nonrobust   LLR p-value:                 4.243e-13
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        -14.9324      9.737     -1.534      0.125     -34.017       4.152
GateDiam      -4.8123      2.709     -1.776      0.076     -10.123       0.498
CupHeight     -0.5460      0.867     -0.630      0.529      -2.245       1.153
SprueHeight   -0.2529      0.663     -0.382      0.703      -1.552       1.046
RiserDiam      5.1473      1.155      4.457      0.000       2.884       7.411
SprueDiamBot  -13.7618      4.284     -3.213      0.001     -22.158      -5.366
SprueDiamTop   -8.8910      2.445     -3.637      0.000     -13.683      -4.099
RiserHeight    0.5067      0.875      0.579      0.562      -1.208       2.221
Riser1Pos     -0.0719      0.839     -0.086      0.932      -1.717       1.573
```

```
Riser2Pos       -0.2144       0.766     -0.280       0.780      -1.716        1.287
==============================================================================
"""
```

The `Logit()` function gives the natural log of the odds that y ("Feasible") equals one of the categories (0 or 1). If we were to use the regular values of y, as opposed to $ln(P(Y = y))$, for the response/dependent variable and tried to fit a line, it wouldn't be a very good representation of the relationship. If you don't believe us, give linear regression a try and see what you find out!

1.6.1 Question 2

Interpret the summary output: Which variables appear to be statistically significant at the 95% level, and how do they affect feasibility? Are the results from the logistic regression consistent with your interpretation of the boxplots?

1.7 Removing non-significant predictors

Let's simplify our model by removing those covariates that are not statistically significant at the 95% level. If variables are not predictive, it is sometimes better to remove them as we can get better predictions from the simpler model when we predict on new data.

1.7.1 Question 3

Change X so that it only includes those covariates that are statistically significant at the 95% level. Fit a new logistic regression model using this dataframe X. Call the fitted model `new_model`. Print out the summary of the new model.

1.8 Multivariate Regression

We will now fit a multivariate regression model for batch time using the same predictors.

Fit a multivariate regression model, using the code below. All mold-variables are used and additional non-linear terms in GateDiam are added. This model was suggested by the analysis in the previous recitation.

Feasible is not used since it is a response, not a mold-variable.

[26]:
```
y = df['BatchTime']
X = df[predictors].copy()
X = sm.add_constant(X)
X['GateDiamSq'] = X['GateDiam']**2
X['GateDiamCube'] = X['GateDiam']**3
X
```

[26]:

	const	GateDiam	CupHeight	SprueHeight	RiserDiam	SprueDiamBot	\
0	1.0	0.500000	5.000000	10.000000	7.000000	0.500000	
1	1.0	0.318317	5.604768	10.148839	6.745247	0.635632	
2	1.0	0.353834	5.846289	9.528183	7.345273	0.642489	
3	1.0	0.596163	4.280124	10.887751	6.952856	0.317180	
4	1.0	0.484338	5.205026	11.003688	7.254450	0.531424	

```
..    …       …         …          …         …          …
95    1.0  0.535138  4.731912    9.851363  6.754550   0.476329
96    1.0  0.255820  5.482461   11.287504  6.179215   0.590127
97    1.0  0.745953  5.008650    9.989357  7.412295   0.624900
98    1.0  0.443939  4.782634   10.488185  7.285994   0.528419
99    1.0  0.633789  4.831297    8.848969  7.668250   0.510992

    SprueDiamTop  RiserHeight  Riser1Pos  Riser2Pos  GateDiamSq  GateDiamCube
0       0.750000     5.000000   3.500000   5.000000    0.250000      0.125000
1       0.889822     5.318354   3.168774   5.233322    0.101326      0.032254
2       1.005683     4.836570   3.265961   5.482437    0.125198      0.044299
3       0.589558     5.492964   4.029533   4.719791    0.355411      0.211883
4       0.595769     4.897376   4.011947   4.822747    0.234583      0.113617
..           …            …          …          …           …             …
95      1.078394     4.773315   3.585070   5.561820    0.286373      0.153249
96      0.726530     5.412255   3.425544   4.578372    0.065444      0.016742
97      0.641878     4.632735   3.977273   5.305690    0.556445      0.415082
98      0.883926     4.503812   3.572636   5.061096    0.197082      0.087493
99      0.999930     4.775548   3.410252   5.492044    0.401689      0.254586

[100 rows x 12 columns]
```

[27]:
```
new_linear_model = sm.OLS(y,X).fit()
new_linear_model.summary()
```

[27]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:              BatchTime   R-squared:                       0.835
Model:                            OLS   Adj. R-squared:                  0.815
Method:                 Least Squares   F-statistic:                     40.52
Date:                Sun, 27 Mar 2022   Prob (F-statistic):           1.01e-29
Time:                        21:23:22   Log-Likelihood:                -309.10
No. Observations:                 100   AIC:                             642.2
Df Residuals:                      88   BIC:                             673.5
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       3028.3393     20.371    148.662      0.000    2987.857    3068.822
GateDiam    -476.1945     90.302     -5.273      0.000    -655.651    -296.738
CupHeight      1.1435      1.519      0.753      0.454      -1.876       4.162
SprueHeight   -0.8335      1.056     -0.790      0.432      -2.931       1.264
RiserDiam      9.5573      1.141      8.374      0.000       7.289      11.825
SprueDiamBot   4.5050      4.838      0.931      0.354      -5.110      14.120
```

```
SprueDiamTop      1.4360      3.480      0.413      0.681      -5.480       8.352
RiserHeight       9.1068      1.381      6.594      0.000       6.362      11.852
Riser1Pos        -0.7958      1.560     -0.510      0.611      -3.895       2.304
Riser2Pos         0.7674      1.263      0.608      0.545      -1.742       3.277
GateDiamSq      694.4074    176.257      3.940      0.000     344.134    1044.681
GateDiamCube   -367.9510    108.946     -3.377      0.001    -584.457    -151.445
==============================================================================
Omnibus:                        0.361   Durbin-Watson:                   1.985
Prob(Omnibus):                  0.835   Jarque-Bera (JB):                0.523
Skew:                          -0.102   Prob(JB):                        0.770
Kurtosis:                       2.710   Cond. No.                     6.14e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 6.14e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

1.8.1 Question 4

What variables are statistically significant at the 95% level in your linear regression? Change X so that it only contains these covariates. Fit a new linear regression model using this dataframe X. Print out the summary of the new model.

1.8.2 Question 5

What is the adjusted R^2 of the old linear regression model? Of the new one? You should see that it has improved. Adjusted R^2 increases when a model makes better predictions on the training data (i.e., the residuals are smaller) and when the model is simpler (i.e., has fewer covariates). By dropping the non-statistically-significant predctors, we obtained a much simpler model without increasing the size of the residuals too much.

Now we will use the logistic model for `Feasible` and the linear model for `BatchTime`, to inform how to set the values of the mold-variables to minimize expected `BatchTime` and maximize $P(Feasible = 1)$.

1.8.3 Question 6

For each of the mold-variables and each of the outcomes $P(Feasible = 1)$ and `BatchTime` (except intercepts, if you included them), consider increasing the variable by a small amount from its mean value in the full dataset. State whether this would increase the outcome, decrease it, or have no effect. Base your answers on the models for each response variable obtained by dropping non-statistically-significant covariates.

Hint: Consider the following plot to when trying to understand the effect of `GateDiam` on `BatchTime`. The plot shows how $GatePoly(X_{GateDiam}) := \beta_{GateDiam} * X_{GateDiam} + \beta_{GateDiamSq} * X_{GateDiamSq} + \beta_{GateDiamCube} * X_{GateDiamCube}$ changes as `GateDiam` increases.

Hint: The logistic function, $L(z)$, is a strictly increasing function. So if $z = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \cdots + \hat{\beta}_j X_j$ increases as you increase X_j, then so does $L(z)$.

[ ]:
```
GateDiamPoly = X[['GateDiam','GateDiamSq','GateDiamCube']].dot(new_linear_model.
 ↪params[['GateDiam','GateDiamSq','GateDiamCube']])
plt.scatter(df['GateDiam'],GateDiamPoly)
plt.xlabel('GateDiam')
plt.ylabel('GateDiamPoly')
```

1.9 Question 7

Our goal is to set the mold-variables to minimize expected `BatchTime` and maximize $P(Feasible = 1)$. We will constrain the value of each predictor to be between the minimum and maximum values in `castdata.csv`.

Based on your answers to the previous question, for each variable that affected one response variable but not the other, state its optimal value for increasing the probability of feasibility and decreasing batch time. For most (and perhaps all) predictors, the value will either be at the minimum or maximum of the values in `castdata.csv` for that predictor — in these cases you can just say `min` or `max`.

2 Randomized Search for Better Operating Conditions

We now search to find values of the mold-variables that give a low expected `BatchTime` subject to the probability that the casting is feasible being at least 0.95. This problem is best solved by linear programming, but, since linear programming is not a prerequisite for this course, we will use a randomized search.

Now we create a new set called newdata with 1,000,000 rows where the predictors vary uniformly between their smallest and largest values in the original data. The responses `BatchTime` and `Feasible` are not included in this data set. Rather, the expected values of `BatchTime` and `Feasible` are predicted using the logistic model and the linear model, respectively.

[ ]:
```
upperbounds = df[predictors].max()
lowerbounds = df[predictors].min()
N = 1000000
np.random.seed(10)
newdata = pd.DataFrame()
for i in range(9):
  newdata[df.columns[i]] = np.random.uniform(lowerbounds[i],upperbounds[i],N)

newdata = sm.add_constant(newdata)
newdata['GateDiamSq']=newdata['GateDiam']**2
newdata['GateDiamCube']=newdata['GateDiam']**3
newdata.head()
```

Next, use `model.predict()` to compute the probability that `Feasible` is equal to 1, and create a sub-dataset called `newdata_feas` of `newdata` where this probability exceeds 0.95.

Note: be careful when using `.predict()` for classification problems. Depending on the package it will give the probability of the outcome being 1 or a prediction (0 or 1). For example when using sklearn.LogisiticRegression, `.predict()` gives a 0-1 prediction and `.predict_prob()` gives the probability of the outcome being 1.

Before you run this code, you will need to fill in the variables you are using in your models from Questions 3 and 5 into `logit_variables` and `linear_variables`. The block of code uses the logistic regression model you fit in Question 3, which should have been called `new_model`.

Hint: if you included an intercept when you fit the model in Q3 (by calling `sm.add_constant` to add a column called `const`), this will cause `new_model` to assume that the dataframes passed to `new_model.predict` will also include this column. Think about whether or not you should include a constant when you answer Q3. If you decide to include it in Q3, include `const` into `logit_variables` here. If you decide not to include it in Q3, then don't include `const` into `logit_variables`.

[ ]:
```
#predict probability of feasible block for each data point and remove points
#with probability less than 0.95
logit_variables = # Fill in the variables from Question 3 here
Yhat = new_model.predict(newdata[logit_variables])
newdata_feas = newdata[pd.Series(Yhat >= 0.95)]
```

Now, predict the value of `BatchTime` for all rows of newdata_feas and find the row where `BatchTime` is minimized; this is the single row of the dataset.

[ ]:
```
linear_variables = # Fill in the variables from Question 5 here
exBatchTime = new_linear_model.predict(newdata_feas[linear_variables])
```

2.1 Question 8

What is the expected value of BatchTime under optimal conditions? "Optimal conditions" means that the predictors are set to the best feasible values in newdata.

[ ]:
```
#your code here
```

To help you answer the next question, here are the minimum and maximum value of each predictor.

[ ]:
```
df[predictors].min()
```

[ ]:
```
df[predictors].max()
```

2.2 Question 9

What are the "optimal" values of the variables, where "optimal" means minimizing the expected `BatchTime` subject to $P(Feasible = 1) > 0.95$. Compare these results with your answer to Question 7 and say whether your results are consistent. You may also want to output the maximum and minimum values for each variable using the code `df.max()` and `df.min()`

Ans:

[ ]:
```
#your code here
```

2.3 Question 10

What is the probability that Feasible will equal 1 under optimal conditions? Hint: Use predict again.

Ans:

[ ]:
```
#your code here
```