Hiatus Code Guidebook

- Lauren Toland
- Michael Gabriel
- Ryan Pittman
- Matthew Sirico
- Joshua Pam
- Daniel De Las Heras

- Professor Venkatesh

ADELPHI
UNIVERSITY
NEW YORK

Data Set Format

	A	B	C	D	E	F	G	H
1		Start Date	End Date	Spend	Impressions	Clicks	Revenue	Network
2	0	2022-02-01	2022-02-07	52586	6660456	122628	69108	Social Media 1
3	1	2022-02-08	2022-02-14	68943	7446126	120792	77064	Social Media 1
4	2	2022-02-15	2022-02-21	46617	5980974	55362	46956	Social Media 1
5	3	2022-02-22	2022-02-28	61807	8360871	96312	70512	Social Media 1
6	4	2022-03-01	2022-03-07	35199	5263032	49827	34320	Social Media 1

For the code to work the data set has to take the following format

Import Libraries

```
[1]: # Import Libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from scipy.stats import norm
import plotly.express as px
import statsmodels.formula.api as smf
from sklearn.linear_model import HuberRegressor
plt.style.use('ggplot')
```

Prior to running the code, you must import all of the Python Libraries used throughout the code.

Creating Data Frame

```
# Create the data frame

path = r"/content/hiatus.csv" # Get the file path from your local machine
df = pd.read_csv(path).drop(columns=['End Date', 'Unnamed: 0']) # The Unnamed: 0 column may not
# be in your data frame, comment out or remove if this is the case
df['Start Date'] = pd.to_datetime(df['Start Date'])
df['Click Through Rate'] = df['Clicks'] / df['Impressions']
df = df.set_index('Start Date')[['Spend','Click Through Rate','Network', 'Revenue']].fillna(0)
sm1 = df[df['Network'] == 'Social Media 1'].drop(columns='Network')
sm2 = df[df['Network'] == 'Social Media 2'].drop(columns='Network')
sm3 = df[df['Network'] == 'Social Media 3'].drop(columns='Network')
search = df[df['Network'] == 'Search'].drop(columns='Network')
```

- Read the file from your local machine with the “r” command and store it in a variable named “path”. Change the file path to where you saved the csv on your local machine.
- Make sure that if you have a column for the end date in your data, you must remove this entire column from your data frame.
- Also, there is a column of index values that comes from converting the excel file into a CSV file. This too must be removed. This is done automatically by the code.
- Create a column for the Click Through Rate (Clicks/Impressions).
- Create separate data frames for each channel.

Binary/Categorical Variables Example

SM1	SM2	SM3	Search
0	1	0	0
1	0	0	0
0	0	0	1
1	0	0	0
0	0	1	0
0	1	0	0
0	0	0	1
0	0	1	0
0	0	0	1
0	1	0	0

- Above is an example of how we accounted for categorical variables using binary values.

Identifying Outliers

```
# List outliers and other anomalies

print('Significant at p < 0.05: \n\n')
outliers = df[df['Revenue'] > df['Revenue'].mean() + 3*df['Revenue'].std()]
outliers['Spend Z-Score'] = (outliers['Spend'] - df['Spend'].mean()) / df['Spend'].std()
outliers['Click Through Rate Z-Score'] = (outliers['Click Through Rate'] - df['Click Through Rate'].mean()) / df['Click Through Rate'].std()
outliers['Revenue Z-Score'] = (outliers['Revenue'] - df['Revenue'].mean()) / df['Revenue'].std()
outliers['Spend p-value'] = norm.sf(outliers['Spend Z-Score'])
outliers['Click Through Rate p-value'] = norm.sf(outliers['Click Through Rate Z-Score'])
outliers['Revenue p-value'] = norm.sf(outliers['Revenue Z-Score'])
outliers = outliers[['Network','Spend','Spend Z-Score','Spend p-value','Click Through Rate', 'Click Through Rate Z-Score',
                     'Click Through Rate p-value', 'Revenue','Revenue Z-Score', 'Revenue p-value']]
outliers
```

- Outliers were determined by considering records where the revenue was more than 3 standard deviations from the mean.
    - line 2 of the code outlines the 3 standard deviation parameter
- Records with a z-score greater than three standard deviation were identified as anomalies and removed.

Turn Network Into Categorical Variable

```
[4]: # Turn Network into a categorical variable

df = df[df['Revenue'] <= df['Revenue'].mean()+3*df['Revenue'].std()] # Remove anomalies
sm1 = df[df['Network'] == 'Social Media 1'].drop(columns='Network')
sm2 = df[df['Network'] == 'Social Media 2'].drop(columns='Network')
sm3 = df[df['Network'] == 'Social Media 3'].drop(columns='Network')
search = df[df['Network'] == 'Search'].drop(columns='Network')
get_dumms = pd.get_dummies(df['Network'])
frames = [df.drop(columns=['Network','Revenue']),get_dumms]
perc_removed = len(df[df['Revenue'] > df['Revenue'].mean()+3*df['Revenue'].std()]) / len(df) * 100
print('Percent of data removed:', str('{0:.2f}'.format(perc_removed))+'%')

Percent of data removed: 1.43%
```

- This part of the code breaks the original data frame into sub columns that allow for analysis to be run on independent social media channels.
- There are methods within this section to also clean the data and return the amount of data removed.
- Note: the number of categories can be expanded upon if Hiatus were to expand marketing onto a new network.

Creating a New Preprocessed Data Frame

```
[5]: # Create new preprocessed data frame

preprocessed = pd.concat(frames,axis=1)
preprocessed['Revenue'] = df['Revenue']
```

The main function of this section regroups the data into a dataframe from the newly cleaned categories.

Split Independent and Dependent Variables

```
[6]: # Split dependent and independent variables

X = preprocessed.drop(columns='Revenue')
y = preprocessed['Revenue']
```

The variables predictor variables (X) and response variable (Y) are assigned their own dataframe feature accordingly.

ADELPHI UNIVERSITY

Finding The Best Split for The Mean Squared Error

```
[7]: # Find the best split for the mean squared error

splits = []

for i in range(1,10):
  splits.append(train_test_split(X, y, test_size=i/10, random_state=42))
```

```
[8]: # Create lists for the training and testing data

X_TRAIN = []
X_TEST = []
y_TRAIN = []
y_TEST = []

for i in range(0,9):
  X_TRAIN.append(splits[i][0])
  X_TEST.append(splits[i][1])
  y_TRAIN.append(splits[i][2])
  y_TEST.append(splits[i][3])
```

- This step is to determine which proportion of the data is best to use for testing the model.
- The minimum MSE is chosen automatically but the code to determine the optimal train/test split for any given dataset.

Performing Huber Regression on Each Split

```
[9]: # Perform Huber regression on each split

HR = []

for i in range(0,9):
  HR.append(HuberRegressor().fit(X_TRAIN[i],y_TRAIN[i]))
```

- Huber regression appears to be the best regressor for the purposes of this analysis.
- This is a regressor that uses L1 and L2 normalization as a piecewise function to minimize the residual terms.

Getting Predictions

```
[10]: # Get predictions

Y_HAT = []

for i in range(0,9):
    Y_HAT.append(HR[i].predict(X_TEST[i]))
```

- Get the predicted values for each testing split.

Getting The Mean Squared Errors

```
[11]: # Get the mean squared errors

MSE = []

for i in range(0,9):
  MSE.append(mean_squared_error(y_TEST[i],Y_HAT[i]))
```

- Determine which split has the minimum MSE.

Creating a Dataframe For The Error Terms

```
# Create a dataframe for the error terms

errors = []
MSE = [mse_1,mse_2,mse_3,mse_4,mse_5,mse_6,mse_7,mse_8,mse_9]
size = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
for i in range(len(MSE)):
    err_dict = {'Test Size' : size[i],
                'MSE' : MSE[i]}
    errors.append(err_dict)
error_df = pd.DataFrame(errors)
```

- Make a dataframe for the error terms used in the plot.

Plotting The Mean Squared Error by Test Size

```
[12]: # Plot the MSE by test size

size = error_df.sort_values(by='MSE').iloc[0][0]
min_err = error_df.sort_values(by='MSE').iloc[0][1]
print('Minimum MSE: ', min_err)
print('Test Size: ', size,'\n')
plt.plot(error_df['Test Size'],error_df['MSE'])
plt.xlabel('Test Proportion')
plt.ylabel('MSE')
plt.title('Mean Squared Error')
plt.show()
```

- Plot the error terms by testing size to visualize the ideal train/test split at the given minimum MSE.

Creating The Model and Fit it to The Training Data

```
[13]: # Create the model and fit to the training data

huber = HuberRegressor()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=size, random_state=42)
huber.fit(X_train,y_train)
```

- After getting the optimal testing proportion we fit the model to this data.

Getting Model Results

```
[14]: # Get model results

sm1_score = r2_score(sm1['Revenue'],huber.predict(preprocessed[preprocessed['Social Media 1']==1].drop(columns=['Revenue'])))
sm2_score = r2_score(sm2['Revenue'],huber.predict(preprocessed[preprocessed['Social Media 2']==1].drop(columns=['Revenue'])))
sm3_score = r2_score(sm3['Revenue'],huber.predict(preprocessed[preprocessed['Social Media 3']==1].drop(columns=['Revenue'])))
search_score = r2_score(search['Revenue'],huber.predict(preprocessed[preprocessed['Search']==1].drop(columns=['Revenue'])))
scores = ['*','*',search_score,sm1_score,sm2_score,sm3_score] # Scores on each channel according to the model
type_ = ['[0,inf)','[0,1]','{0,1}','{0,1}','{0,1}','{0,1}']
category = ['Numeric','Numeric', 'Categorical','Categorical', 'Categorical', 'Categorical']

huber_dict = []

for i in range(len(huber.coef_)):
  dictionary = {'Variable' : preprocessed.columns[:-1][i],
                'Type' : category[i],
                'Index (i)' : i+1,
                'Domain (Xi)' : type_[i],
                'Coefficient (βi)' : huber.coef_[i],
                'R-Squared Score: ': scores[i]}
  huber_dict.append(dictionary)
huber_df = pd.DataFrame(huber_dict).set_index('Index (i)')
```

- Create a dataframe describing the results and parameters for the model.

Printing Model Statistics For Social Model 1,2,3 and Search

```
# Summary Statistics for Social Media

print('Social Media 1 Summary Statistics: \n\n')
sm1_stats = sm1.describe().T
sm1_stats
```

```
# Summary Statistics for Social Media 2

print('Social Media 2 Summary Statistics: \n\n')
sm2_stats = sm2.describe().T
sm2_stats
```

```
# Summary Statistics for Social Media 3

print('Social Media 3 Summary Statistics: \n\n')
sm3_stats = sm3.describe().T
sm3_stats
```

```
# Summary Statistics for Search

print('Search Summary Statistics: \n\n')
search_stats = search.describe().T
search_stats
```

- Use the print function to describe the summary statistics (count, mean, standard deviation, minimum, maximum, interquartile range).
- Repeat this individually for each network.

Plotting Results With Randomized Test Data

```
[19]: # Plot results with randomized test data

      plt.plot(huber.predict(X_test),color='r',label='Predicted', linewidth=1)
      plt.title('Huber Regression on Randomized Test Data')
      plt.plot(y_test.values,color='k', label='Actual', linewidth=1)
      plt.legend()
      plt.ylabel('Revenue')
      plt.xlabel('Record')
      plt.show()
```

- This step is to visualize how well the model fits the testing data.

Statistics of The New Huber Regression

```
[21]: # Compare with the new Huber Regressor

yhat = np.array(huber.predict(X_test)).reshape(-1,1)
print('Huber Regression Results: \n\n')
print('Target (Y): ', 'Revenue')
print('Model: ', 'Huber Regressor')
print('R-Squared Score: ', huber.score(X_test,y_test))
print('Test Size: ', size)
print('Mean-Squared Error: ', min_err)
print('Bias (β0): ', huber.intercept_) # Number added to the model
print('Scale: ', huber.scale_)
print('Parameters: ', huber.get_params(),'\n')
huber_df
```

- The R-squared column describes how much of the variance the model describes for each channel individually.
- The domain column describes the range of inputs the model will accept for each dependent variable. The domain will specify what value bounds spend and click through rate can take. The categorical variables associated with the individual channels are binary.
- Domain could be 0 to 1 (click through rate-proportion)
- Domain could be 0 to infinity (spend)
- Domain is range of values that are appropriate, for example spend could not be negative

Allocation of The Budget

```
[22]: allocation = []
      total = 0
      my_list = list(huber.coef_[2:6])
      for i in my_list:
        if i<0:
          my_list.remove(i)
          my_list.append(0)
      for i in my_list:
        total += i
      for j in my_list:
        new = j/total
        allocation.append(new)
```

```
[23]: channel = ['Search', 'Social Media 1', 'Social Media 2', 'Social Media 3']
      allocation_data = []

      for i in range(len(channel)):
        alloc_dict = {'Channel' : channel[i],
                      'Percent Spread' : str("{:.2f}".format(allocation[i]*100))+'%'}
        allocation_data.append(alloc_dict)
      allocation_df = pd.DataFrame(allocation_data).set_index('Channel')
      allocation_df.loc['Social Media 3'][0] = '<1%'
      allocation_df
```

- If we turn the coefficients of the model into a proportion, assuming all other variables remain constant, then this will give the optimal spread for each channel.
- These proportions are the percentages of suggested capital allocation for each ad platform.

Coefficients

```
Huber Regression Results:


Target (Y):  Revenue
Model:  Huber Regressor
R-Squared Score:  0.8402630771805806
Test Size:  0.7
Mean-Squared Error:  601862310.730497
Bias (β0):  13087.678733001265
Scale:  9444.008181731271
Parameters:  {'alpha': 0.0001, 'epsilon': 1.35, 'fit_intercept': True, 'max_iter': 100, 'tol': 1e-05, 'warm_start':
```

Index (i)	Variable	Type	Domain (Xi)	Coefficient (βi)	R-Squared Score:
1	Spend	Numeric	[0,inf)	1.18	*
2	Click Through Rate	Numeric	[0,1]	389.07	*
3	Search	Categorical	{0,1}	7484.43	0.258802
4	Social Media 1	Categorical	{0,1}	409.96	0.684478
5	Social Media 2	Categorical	{0,1}	17778.01	0.776334
6	Social Media 3	Categorical	{0,1}	-14425.16	0.935177

ADELPHI UNIVERSITY

Coefficients

	Coefficients	Totals	Percentages	Proportion of $0.18	Convert	Final Spread
Click-Through-Rate	389.07	389.07	0.009609839	0.001729771	0.00173	----------------
Search	7484.43	7484.43	0.184861768	0.033275118	5.8E-05	0.033332677
SM1	409.96	409.96	0.010125812	0.001822646	3.2E-06	0.001825799
SM2	17778.01	17778.01	0.43910817	0.079039471	0.00014	0.079176191
SM3	-14425.16	14425.16	0.356294411	0.064132994	0.00011	0
		40486.63		0.18		0.114334666

```
[86] # Make coeffiecients interpretable, determine the magnitude of each platform on $0.18 return.

     coefs = huber_df['Coefficient (βi)'][1:].astype('float').values
     coefs[-1] = abs(coefs[-1])
     coef_sum = coefs.sum()
     cents = float(huber_df['Coefficient (βi)'][1]) - 1
     freq = ((coefs / coef_sum) * cents)[1:]
     rel_freq = (coefs / coef_sum) * cents
     pop = rel_freq[0]
     rel_freq = rel_freq[1:]
     sol = rel_freq * pop
     solution = (freq + sol)[:-1]
```

ADELPHI UNIVERSITY

Coefficients

```
# Store new dataframe

solution_df = pd.DataFrame()
solution_df['Channel'] = ['Search', 'Social Media 1', 'Social Media 2']
solution_df['Contribution'] = solution
solution_df = solution_df.set_index('Channel')
solution_df
```

Channel	Contribution
Search	0.033333
Social Media 1	0.001826
Social Media 2	0.079176

```
[88] # Overall return on the dollar when SM3 has zero spend

cont = round(solution_df['Contribution'].sum(),2)
cont
```

```
0.11
```