CA270 Project
Ronan Kelly and Darragh Nagle

Section One - Idea, Data description, any form of data preparation.

Idea generation - We both agreed that it would be best if we focused on something we share a common interest in. Therefore, we chose to base it on soccer (Football).

Of course there is a lot within this sport so we had to narrow it down even more.

We originally thought of predicting the upcoming PL (Premier League) season, but with informative feedback from our lecturer we found out this task would be too hard to tackle.

So back to the drawing board we went.

We found it hard to agree on an idea as we didn't really know at that point how we would go about all of the predicting nonsense.

After some time of throwing ideas around, we settled.

We would be using last year's PL goalkeeping stats to train an algorithm to predict how many goals a keeper would concede over a selected few matches

This in theory sounds easy enough, but sports isn't based on stats.
That we learned quickly.

Dataset description - Our dataset is based on the most important goalkeeping statistics in our opinion.

- Save Percentage,
- xG against,
- Clean Sheets
- Post-Shot xG
- Saves
- Shots on Target Against
- Goals Against

We decided to pick 11 goalkeepers from the Premier League which played a huge role in their team to truly see if we could predict how the best of the best goalkeepers would do.

Are main inspiration was the young Irish southampton goalkeeper Gavin Bazunu, but sadly he did not play in the PL last year.

Moment of silence for the lack of Gavin Bazunu data… * insert silence *....

The goalkeepers we eventually decided to analyse were:

- Hugo Lloris
- Allison
- Jordan Pickford
- Emi Martinez
- Jose Sa
- Illan Meslier
- Edouard Mendy
- Ederson
- David De Gea
- Aaron Ramsdale
- Kasper Schmeichel

Data preparation - We knew we could get all the data we needed from (https://fbref.com/).

So we did.

We ran into a slight problem and that was the fact that we could only download data for each keeper separately.

This meant that we had to individually download the data for the selected goalkeepers separately and combine it in one spreadsheet to allow us to run tests on the data as a whole.

Another issue we found is that we only wanted a select amount of rows which influence the amount of goals a keeper would concede, leading to further time spent purely preparing the data set to be worked on.

This is an example of the first 9 rows of our dataset:

	Date	Venue	Result	Squad	Opponent	SoTA	GA	Saves	Save %	CS	PSxG	Opposition XG
Kasper Schmeichel	**2021-08-14**	Home	**W 1–0**	Leicester City	Wolves	3	0	3	100	1	0.3	1.1
Kasper Schmeichel	**2021-08-23**	Away	**L 1–4**	Leicester City	West Ham	7	4	3	42.9	0	3.4	2.5
Kasper Schmeichel	**2021-08-28**	Away	**W 2–1**	Leicester City	Norwich City	3	1	3	100	0	1.1	1.6
Kasper Schmeichel	**2021-09-11**	Home	**L 0–1**	Leicester City	Manchester City	8	1	7	87.5	0	1.3	2.9
Kasper Schmeichel	**2021-09-19**	Away	**L 1–2**	Leicester City	Brighton	3	2	2	66.7	0	1.1	1.4
Kasper Schmeichel	**2021-09-25**	Home	**D 2–2**	Leicester City	Burnley	3	2	2	33.3	0	0.4	0.7
Kasper Schmeichel	**2021-10-03**	Away	**D 2–2**	Leicester City	Crystal Palace	4	2	2	50	0	0.8	1.3
Kasper Schmeichel	**2021-10-16**	Home	**W 4–2**	Leicester City	Manchester Utd	6	2	4	66.7	0	0.7	1.4

The rest of our data can be found through this link-Goalkeeper Stats

We removed all the columns containing strings when doing the algorithm so the numbers were all that affected the prediction.

Section Two - Algorithm description
We first began by doing some research on how to build a predictive algorithm, and what predictive models would be best for predicting goals conceded.

We found this article Linear Regression model.. It had a similar focus to us, as it was trying to find what statistic was best for predicting goals scored. We decided we would attempt to build our own Linear Regression model for our own predictions.

```
[1]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn import model_selection
     from sklearn.linear_model import LogisticRegression
     from sklearn.linear_model import LinearRegression
     from sklearn.metrics import mean_squared_error
     from sklearn.model_selection import train_test_split
     from sklearn.feature_selection import SelectKBest
     from sklearn.feature_selection import chi2
     from sklearn.metrics import r2_score
```

```
[2]: df= pd.read_excel('Linear Regression Algorithm/test-data.xlsx')
     df=df.iloc[0:,:6]
     x=(df.head(39))
     y=[2,2,2,2,1]
     print(df)
```

```
    SoTA  GA  Saves  Save%  CS  PSxG
0      3   0      3  100.0   1   0.3
1      7   4      3   42.9   0   3.4
2      3   1      3  100.0   0   1.1
3      8   1      7   87.5   0   1.3
4      3   2      2   66.7   0   1.1
..   ...  ..    ...    ...  ..   ...
73     5   0      5  100.0   1   1.1
74     4   1      3   75.0   0   1.5
75     3   1      2   66.7   0   1.4
76     3   1      2   66.7   0   0.6
77     2   1      1   50.0   0   0.8

[78 rows x 6 columns]
```

```
[3]: columns = df.columns.tolist()
     target = "GA"
     # Generate the training set.  Set random_state to be able to replicate results.
     train = df.loc[6:38]
```

```
     # Select anything not in the training set and put it in the testing set.
     test = df.loc[2:5]
     # Print the shapes of both sets.
     print("Training set shape:", train.shape)
     print("Testing set shape:", test.shape)
```

```
Training set shape: (33, 6)
Testing set shape: (4, 6)
```

```
]: lin_model = LinearRegression()
   # Fit the model to the training data.
   lin_model.fit(train[columns], train[target])
   # Generate our predictions for the test set.
   lin_predictions = lin_model.predict(test[columns])
   print("Predictions:", lin_predictions)
   # Compute error between our test predictions and the actual values.
   lin_mse = mean_squared_error(lin_predictions, test[target])
   print("Computed error:", lin_mse)
```

```
Predictions: [1. 1. 2. 2.]
Computed error: 1.6220952363607055e-29
```

```
]:
```

We noticed that the algorithm wasn't particularly accurate and so we decided to change to a different predictive model for analysing our data.

We next tried a Naive Bayes predictive model, as it was the model we had learned about at the beginning of this module. A Gaussian Naive Bayes classifier we found to be the best and easiest to work with.

Our first attempt at a Naive bayes model:

```
: import pandas as pd
  from sklearn.model_selection import train_test_split
  from sklearn.naive_bayes import GaussianNB
```

```
: df= pd.read_excel('naive_bayes_algorithm/test-data.xlsx')
  df=df.iloc[: , [7,8,9,10,11,12]]
  print(df.head(2).to_string())
  target= df.GA
  inputs=df.drop('GA', axis="columns")
  x_train, x_test, y_train, y_test = train_test_split(inputs, target, test_size=0.
   ↪2)
```

```
   Saves  Save%   CS  PSxG  Opposition XG   GA
0    3.0  100.0  1.0   0.3            1.1  0.0
1    3.0   42.9  0.0   3.4            2.5  4.0
```

```
: model=GaussianNB()
  model.fit(x_train,y_train)
  print(model.score(x_test,y_test))
  print(y_test)
  print(model.predict(x_test))
```

```
0.7283950617283951
174    1.0
343    1.0
317    3.0
106    0.0
347    0.0
      ...
175    1.0
146    0.0
196    7.0
50     0.0
292    2.0
Name: GA, Length: 81, dtype: float64
[1. 1. 2. 0. 0. 1. 0. 0. 1. 1. 3. 3. 3. 3. 0. 0. 3. 3. 1. 2. 3. 3. 0. 1.
 1. 1. 2. 1. 0. 0. 0. 0. 1. 1. 1. 0. 2. 2. 3. 1. 1. 2. 1. 1. 1. 0. 1. 3. 0.
 0. 0. 1. 0. 1. 0. 1. 1. 2. 2. 1. 0. 1. 0. 1. 0. 0. 1. 1. 1. 3. 0. 1. 0.
```

```
1. 0. 0. 4. 1. 0. 4. 0. 1.]
[ ]:
```

This model came out with an accuracy of between 70-80% and although this model is significantly more favourable and accurate than the Linear Regression model, we knew we could improve this accuracy even further.

Our next and final model was the best so far:

FInal Model

First we imported all the necessary packages.

Side Note: We left in 'Print()', so you can understand what we saw.
This hopefully will help you understand our process.

```
import inline as inline
import pandas as pd
import openpyxl as op
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import category_encoders as ce
import warnings
import statistics

warnings.simplefilter(action='ignore', category=FutureWarning)
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import RobustScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

After this we had to of course import our dataframe.
So this is what is seen below.

We also printed the dimensions of the dataframe to see what we were working with.

Then we separated our database on its column types - Categorical and Numerical.

```
# initializing df
df = pd.read_excel("..\Downloads\gk_stats (1).xlsx")
#print(df.head(1).to_string())


# Dimensions of df
#print(df.shape)


# iloc controls which rows are used.
set_row = df.iloc[0:2]
#print(set_row.to_string())


# Getting categorical columns
categorical = [var for var in df.columns if df[var].dtype == 'O']
#print('There are {} categorical variables\n'.format(len(categorical)))
#print('The categorical variables are :\n\n', categorical)
#print(f'\n{df[categorical].isnull().sum()}')


# Getting numerical columns
numerical = [var for var in df.columns if df[var].dtype != 'O']
#print('There are {} numerical variables\n'.format(len(numerical)))
#print('The numerical variables are :\n\n', numerical)
```

We noticed that some rows under 'Save%' had missing values.
This was due to them not making any saves therefore they had no 'Save%', so we replaced the NA with 0.0.

Also we had to remove date, as it wasn't considered a string, int or float.
This just made our lives easier.

Next was declaring what we were predicting.
Which was 'GA' - Goals Against.

We used to help us ‘train_test_split()’.
The ‘test_size’ determined what percentage of our dataset we would use as the test set.
We settled at 20%.

Once again our data was separated, this time to use for our training set.

Next was one hot encoding.

All of this was helping us build a better algorithm.

```
# Replacing N/a in save% with 0.0 and dropping date
df = df.fillna(0.0)
df = df.drop(['Date'], axis=1)


# Declare feature vector amd target variable
X = df.drop(['GA'], axis=1)
#print(X)
y = df['GA']
#print(y)


# Spliting Data into sep training sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
#print(X_train)
#print(X_test)


# Getting Categorical/numerical columns in training set
#print(X_train.dtypes)
categorical = [col for col in X_train.columns if X_train[col].dtypes == 'O']
#print(f'Categorical:\n{categorical}')
numerical = [col for col in X_train.columns if X_train[col].dtypes != 'O']
#print(f'\nNumerical:\n{numerical}')


# encode remaining variables with one-hot encoding
encoder = ce.OneHotEncoder(cols=['Name', 'Venue', 'Result', 'Squad', 'Opponent'])
X_train = encoder.fit_transform(X_train)
X_test = encoder.transform(X_test)
```

Next was where the fun started.

We trained our dataframe using the function ‘GaussianNB()’.

Then we printed our accuracy. - Which was always around 93%.

```
Model accuracy score: 0.9375
Training-set accuracy score: 0.9969
```

```
# Feature Scaling
cols = X_train.columns
scaler = RobustScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train = pd.DataFrame(X_train, columns=[cols])
X_test = pd.DataFrame(X_test, columns=[cols])
#print(f'{X_train.head(2).to_string()}\n\n')
#print(f'\n\n{y_train.head(2).to_string()}')


# Training our df
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# Predicting results
y_pred = gnb.predict(X_test)
#print(y_test.head(5).to_string())
#print(y_pred)
print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
y_pred_train = gnb.predict(X_train)
#print(y_pred_train)
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))
```

I messed around with the Null accuracy, but didn’t spend too much time on this as I felt it wasn't that important.

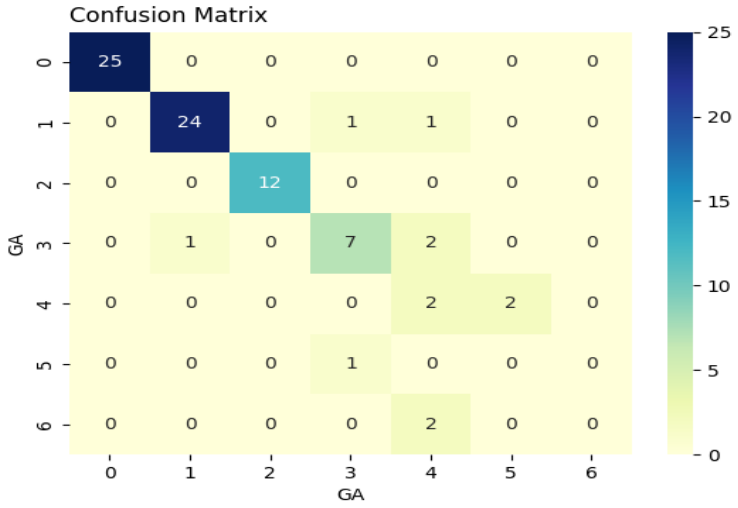
The confusion matrix was useful and helped us understand our result better.

Below the code is one of the many confusion matrices we got.

```
# Comparing to NUll accuracy
#print(y_test.value_counts())
#null_accuracy = (31 / (31 + 27 + 14 + 5 + 3))
#print('Null accuracy score: {0:0.4f}'. format(null_accuracy))


# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
#print('Confusion matrix\n\n', cm)
cm_matrix = pd.DataFrame(data=cm)
heat = sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
#print(classification_report(y_test, y_pred))
```

The values on the diagonal are the ones we correctly predicted.

As you can see, when it came to predicting a keeper conceding 3 >= goals.
We were pretty good at getting it right. But after 3 goals, well you can see.



Section Three - Results and Analysis

Graphing.
This was not needed, but fun.
We thought it could be useful to see what our data gave us.

Simply messing around gives us some cool graphs.

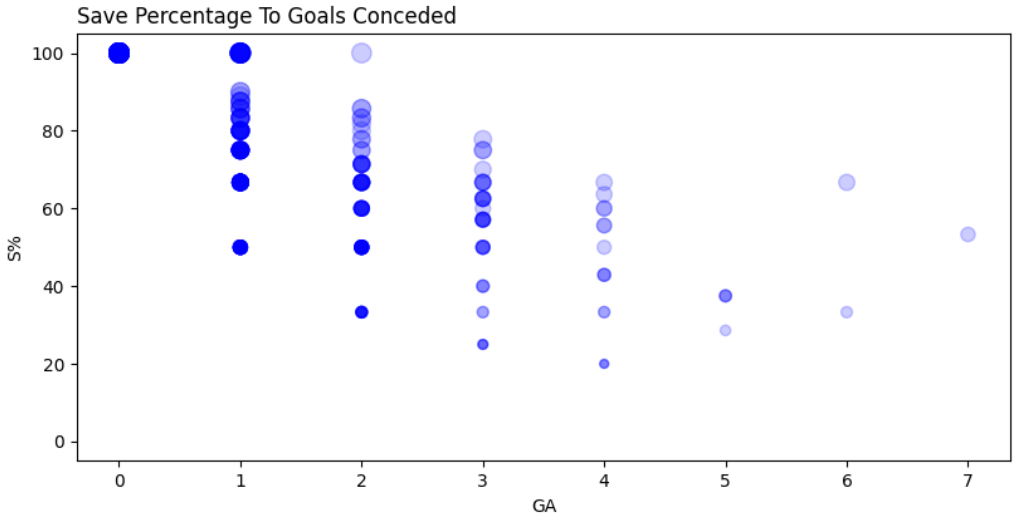
```
#Graphing

x_ax = df['GA']
y_ax = df['Save%']
a, b = np.polyfit(x_ax, y_ax, 1)
m_1 = statistics.mean(x_ax)
sd_1 = statistics.stdev(x_ax)

m_2 = statistics.mean(y_ax)
sd_2 = statistics.stdev(y_ax)

plt.scatter(x_ax, y_ax, s=y_ax*1.25, alpha=.2, color='blue')
#plt.plot(x_ax, a*x_ax+b, color='Green', linestyle='-', linewidth=2, alpha=0.5)
plt.title('Save Percentage To Goals Conceded', loc='left')
plt.ylabel('S%')
plt.xlabel('GA')

edit_df = df.drop(['Name', 'Venue', 'Result', 'Squad', 'Opponent', 'Save%'], axis=1)

edit_df = edit_df.cumsum()
#print(edit_df)
edit_df.plot()

ag = edit_df.drop(['CS', 'SoTA', 'Saves'], axis=1)
ag.plot()

shot = edit_df.drop(['GA', 'CS', 'PSxG', 'Opposition XG'], axis=1)
#SperS = (sum(edit_df['SoTA']))/(sum(edit_df['Saves']))
#SperS_line = range(0, 400, 1)
#S_line = []
#total = 0
#for i in SperS_line:
```

This was the first graph produced.

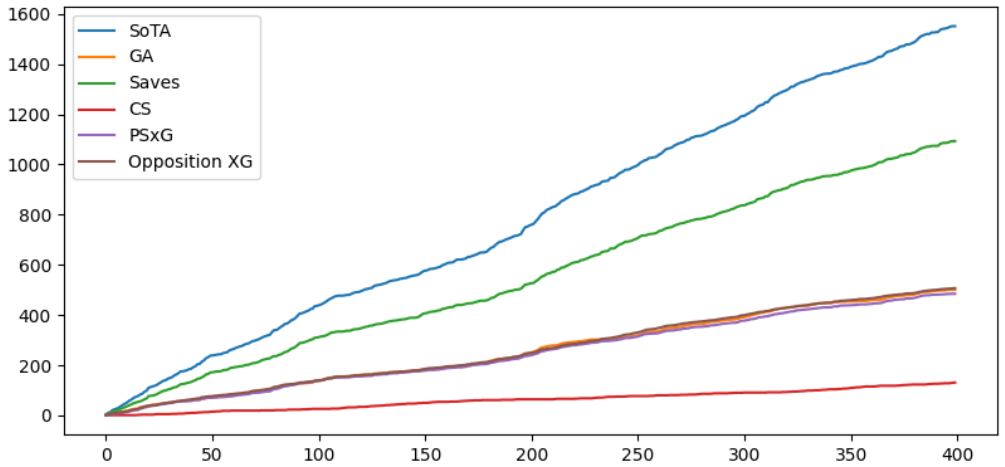
We wanted to see if there was a strong correlation between ‘Save%’ and ‘GA’.
But it wasn’t as dramatic as we thought it would be.

Of course it slowly got worse the more goals the keeper conceded.

Surprisingly, the worst save percentage was not when a goalkeeper conceded the most goals. It occurred when a goalkeeper conceded 4 goals. More than one keeper did this. So maybe if those keepers let in one more their save percentage would go up.

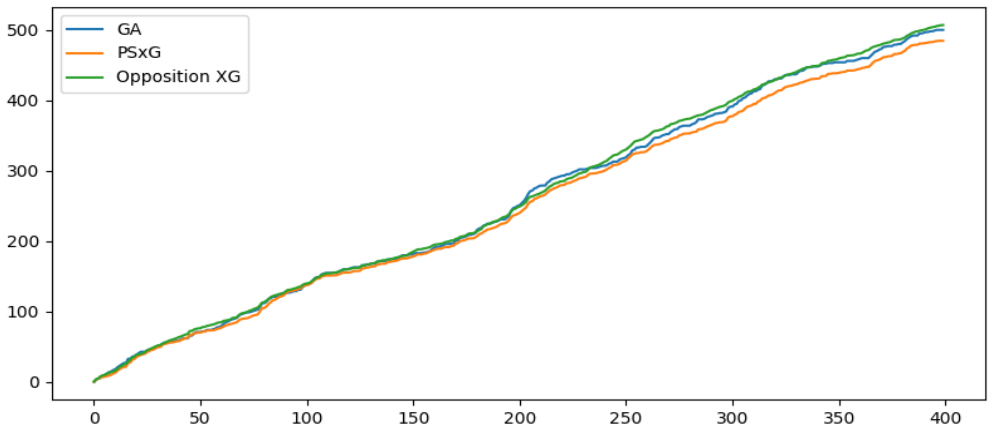


If we graph all the numerical data, game by game, we can see that some stats grow much quicker than others. Therefore we broke this graph into smaller ones.

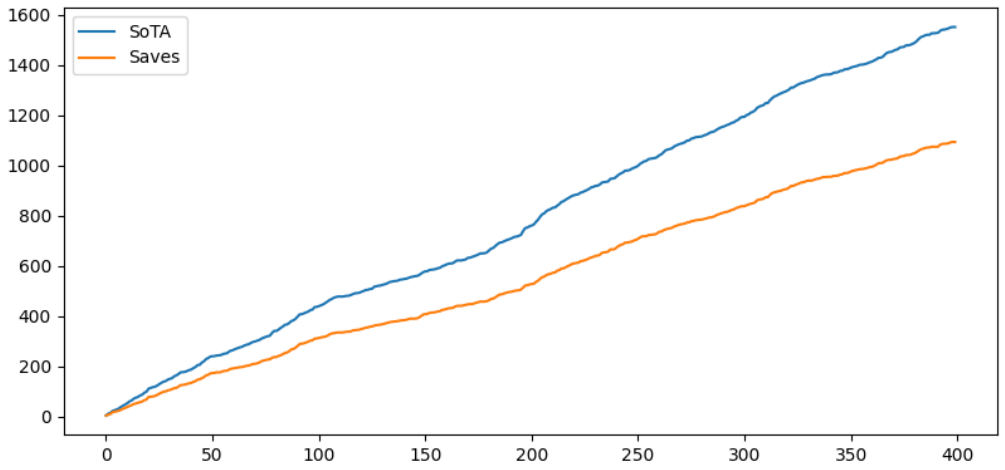


We saw ‘GA’, ‘PSxG’ and ‘Opposition XG’ stuck side by side over the 400 game, which is expected but overall our keepers conceded slightly less goals than expected. This does make sense as the goalkeepers whose data we selected were the best in the world so you would expect them to outperform the average.

This wasn’t also the case during games 200 - 250. Our goalkeepers, as a collective were underperforming, and this resulted in them letting in more goals than expected.



We wanted to compare shots on target against and saves made to see how close these two were by the end of the 400 games. As one may expect, the more games played the bigger the gap between the two.



This was simply down to conceding more goals, the more shots you face.

We found out that it's very hard to predict individual games. But it may be easier to predict a time period. Or so we thought.

Take the graph where we show how many goals a goalkeeper concedes compared to the xG(Expected Goals).

We can see that no 50 game period is the same as goalkeepers perform differently each game.

This may be caused by many things.

And by comparing our graphs we saw that at the same period there is a spike in shots faced. This can mean many different things: Maybe the goalkeepers weren't expecting players to shoot this much and it caught them off guard.

Also simply by reading our graph we can see that the clean sheet rate is pretty consistent. Nice.

We also learned that there are many ways to do the exact same thing. We could have done most if not all of this project within a different application, and many applications where used, such as Gitlab to collaborate, excel, openrefine to Pycharm.

Our data wasn't tricky, in fact it was quite easy to work with beside the date. But once that was gone our data was good to go.

This project was exciting as it was new to us. This meant we had a lot of mistakes to make and we did make mistakes. But this improved our understanding of the data.

By the end of this, we felt very confident in using our dataframe as we knew what was what and what we needed to do.

We look forward to doing something similar within the future as it is enjoyable to tackle this project and to work as a pair.

We have linked our Gitlab repository at the bottom of this report so you are able to look at our different test models and run the code.

Darragh Nagle and Ronan Kelly.

References

Linear Regression Model Article-
https://backofthenetofficial.com/2019/08/05/analysis-what-are-the-best-predictors-of-goals-scored/#:~:text=According%20to%20Opta%20Sports%2C%20%E2%80%9CExpected,best%20predictor%20for%20goals%20scored.

Goalkeeper Stats Spreadsheet-
https://docs.google.com/spreadsheets/d/1hjOioRM990E50bjjhuT73_lgtum50DR4/edit#gid=330429134

Gitlab Repository-
https://gitlab.computing.dcu.ie/kellyr96/ca270_project