Dublin Business School

Student Name	Peterson Donada
Student Number	10521646
Module Title	Machine Learning
Module Code	B9DA104
Lecturer Name	Abhishek Kaushik
Course Title	Master of Science (MSc) in Data Analytics
Assignment Title	CA 2
Word Count	
Data of Submission	20/11/2020

GitHub Regression code:
https://gist.github.com/pdonada/8f09a74a887b7b7a28952cbb410500f5
GitHub Classification code:
https://gist.github.com/pdonada/6cacdcf8d1091ef9bfdcf3e5fc116306

Q1 Define

Data Sampling

Is a technique used to select, manipulate and analyze a representative subset of data points to identify patterns and trends in the larger data set being examined.
Steps:
1. Identify the target population: group of individuals or objects to which researchers are interested in generalizing their findings. For example, research data warehouse use in the banking sector.
2. Select a sampling frame: a list of the population from which the researcher selects units. For example, from the research above select the 4 big banks in Ireland.
3. Specify the sampling technique: sampling can be done through probability (random selection) or non-probability (non-random) technique.
4. Determine the sample size: the sample size is simply the number of units in the sample.
5. Execute the sampling plan: once population, sampling frame, sampling technique and sample size are identified, the analysis take place.

Types of data sampling methods

- probability, using random numbers that correspond to points in the data set to ensure that there is no correlation between points chosen for the sample
- nonprobability, a data sample is determined and extracted based on the judgment of the analyst.

Probability

- Simple random sampling
- Stratified sampling
- Cluster sampling
- Multistage sampling
- Systematic sampling

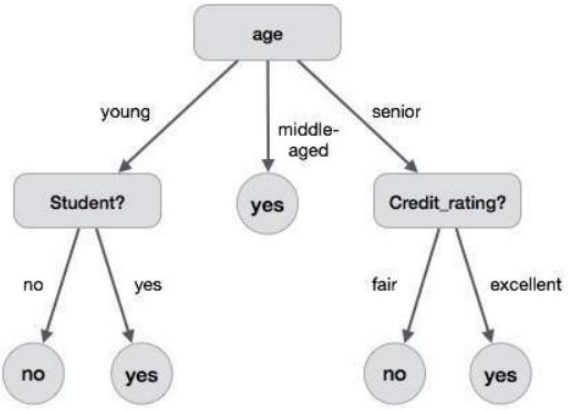
Nonprobability

- Convenience sampling

- Consecutive sampling
- Purposive or judgmental sampling.
- Quota sampling

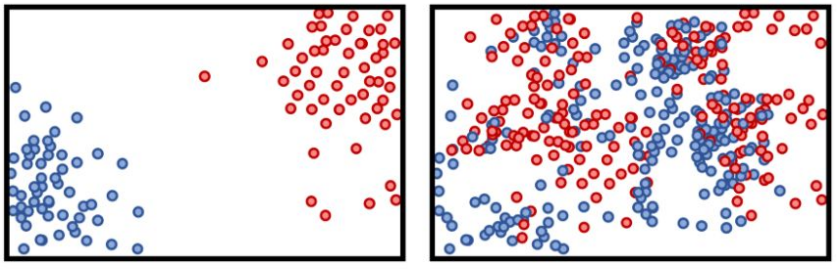
Decision tree

A decision tree is a structure that includes a root node, branches, and leaf nodes. Each internal node denotes a test on an attribute, each branch denotes the outcome of a test, and each leaf node holds a class label. The topmost node in the tree is the root node.



Entropy

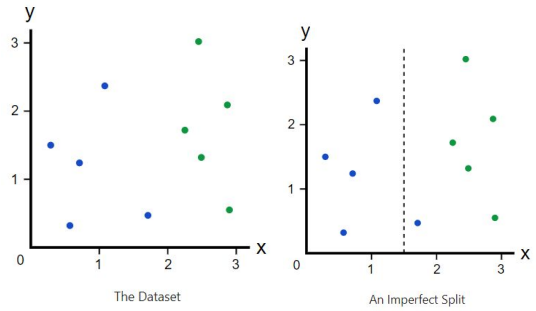
Entropy measures the homogeneity of a dataset. Entropy controls how a Decision Tree decides to split the data. It actually affects how a Decision Tree draws its boundaries.



Information gain

Is a metric used to train Decision Trees. Specifically, these metrics measure the quality of a split. It measures the reduction in entropy or surprise by splitting a dataset according to a given value of a random variable. This is the concept of a

decrease in entropy after splitting the data on a feature. The greater the information gain, the greater the decrease in entropy or uncertainty.



What if we made a split at x = 1.5x=1.5? This imperfect split breaks our dataset into these branches: Left branch, with 4 blues, Right branch, with 1 blue and 5 greens

Chinese restaurant algorithm

A simple way to explain the Chinese restaurant process is that in an imaginary Chinese restaurant with infinite tables, people will cluster at these tables according to a given set of probabilities used by the algorithm. Then, the algorithm will model how many people will sit at each table, in which the tables are the "partitions." The randomization or probabilistic aspect of the Chinese restaurant process can be shown in mathematical form.

In a very general sense, these stochastic processes seek to model human behavior, the behavior of masses of humans, in ways that build enterprise intelligence and direct decision-making.

Agglomerative Hierarchical Clustering

Clustering is basically a technique that groups similar data points such that the points in the same group are more similar to each other than the points in the other groups(cluster).

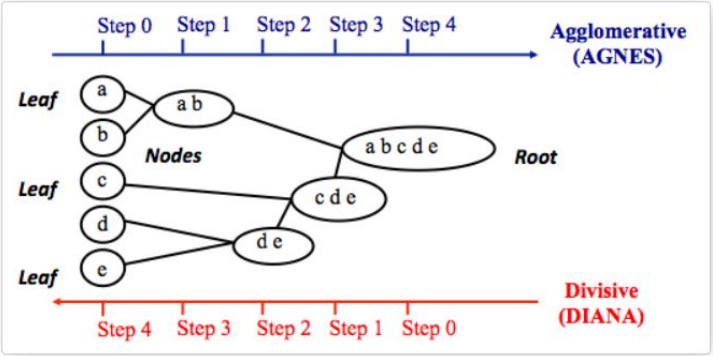
Hierarchical clustering is one of the most popular and easy to understand clustering techniques. It is divided into two types:

- Agglomerative: initially each data point is considered as an individual cluster. At each iteration, the similar clusters merge with other clusters until one cluster or K clusters are formed.
- Divisive: it is exactly the opposite of the Agglomerative. We consider all the data points as a single cluster and in each iteration, we separate the data points from the cluster which are not similar. Each data point which is

separated is considered as an individual cluster. In the end, we'll be left with n clusters.

There are certain approaches which are used to calculate the similarity between two clusters:

- MIN
- MAX
- Group Average
- Distance Between Centroids
- Ward's Method



Q2 Program

Linear Regression

Linear regression is a type of predictive analysis. The overall idea of regression is to examine rather:

- does a set of predictor variables do a good job in predicting an outcome or dependent variable?
- which variables in particular are significant predictors of the dependent variable, and in what way they indicated by the magnitude and sign of the beta estimates impact the outcome variable? These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables.

Three major uses for regression analysis are

1. determining the strength of predictors,
2. forecasting an effect, and
3. trend forecasting.

Types of Linear Regression used in this work:

- Simple linear regression: 1 dependent variable, 1 independent variable
- Multiple linear regression: 1 dependent variable, 2+ independent variables

Database

This dataset describes how Americans felt about Trump's impeachment and the possibility of removing him from office. There are three groups of data: Democrats sample, Republicans sample and Independent supporters.

For this dataset Linear Regression and Multiple Linear Regression will be used. The first to calculate the missing values ("NaN) for the columns "DemYes" and "RepYes", respectively for Democrats and Republicans to vote for the president's impeachment.

The last to calculate a predictive of the "Yes" column, which represents the support of impeachment for independent supporters, based on Democrats and Republicans "Yes" volte.

Data Inspection

The database is a pandas dataframe with 550 lines and 24 columns from which a subset of 6 columns will be extracted to be analyzed: 'Yes', 'No', 'Rep Yes', 'Rep No', 'Dem Yes', 'Dem No'.

A subset was created with the features to be used:

```
In [17]: data_sub.head()
Out[17]:
    Yes    No  RepYes  RepNo  DemYes  DemNo
0  37.0  59.0     7.0   87.0    61.0   36.0
1  37.0  56.0    10.0   87.0    62.0   29.0
2  40.0  55.0     7.0   90.0    64.0   30.0
3  49.0  46.0    15.0   82.0    75.0   21.0
4  43.0  51.0     5.0   93.0    77.0   15.0
```

There are missing values for two features which will be used in the multivariate regression and these will be imputed using Linear Regression: “DemYes” and “RepYes”.

```
In [16]: data_sub.isnull().sum(axis=0)
Out[16]:
Yes        0
No         0
RepYes    33
RepNo     50
DemYes    27
DemNo     51
dtype: int64
```

Correlations

Before starting the linear regression let's check the correlations. The strength of the linear association between two variables is quantified by the correlation coefficient.The correlation coefficient always takes a value between -1 and 1, with 1 or -1 indicating perfect correlation (all points would lie along a straight line in this case). A positive correlation indicates a positive association between the variables (increasing values in one variable correspond to increasing values in the other variable), while a negative correlation indicates a negative association between the variables (increasing values is one variable corresponds to decreasing values in the other variable). A correlation value close to 0 indicates no association between the variables.

[9]:
```
# correlations
data_sub.corr()
```

[9]:

	Yes	No	RepYes	RepNo	DemYes	DemNo
Yes	1.000000	-0.429575	0.333139	-0.142469	0.874976	-0.728369
No	-0.429575	1.000000	-0.456485	0.707334	-0.350757	0.716131
RepYes	0.333139	-0.456485	1.000000	-0.796774	0.000592	-0.075146
RepNo	-0.142469	0.707334	-0.796774	1.000000	0.136618	0.155845
DemYes	0.874976	-0.350757	0.000592	0.136618	1.000000	-0.838131
DemNo	-0.728369	0.716131	-0.075146	0.155845	-0.838131	1.000000

Data Skew

Data skew primarily refers to a non uniform distribution in a dataset. Skewed distribution can follow common distributions (e.g., Zipfian, Gaussian, Poisson).

Negative Skew: the long "tail" is on the negative side of the peak or to the left.

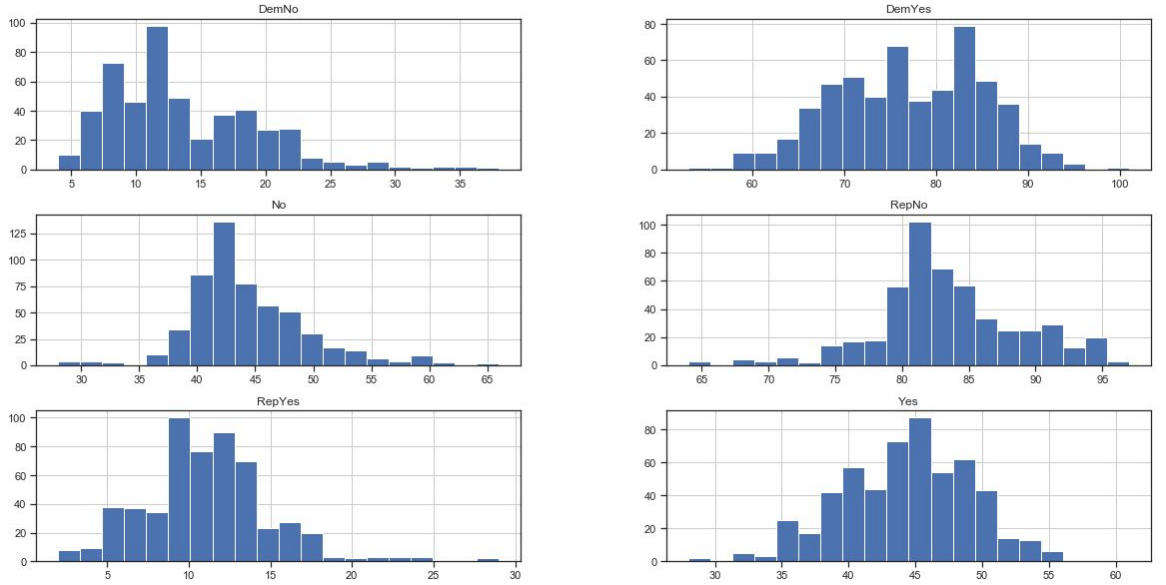
Positive Skew: the long tail is on the positive side of the peak or to the right.

Normal Distribution: is not skewed but symmetrical and the mean is at the peak.

[10]:
```
# skew
data_sub.skew()
```

[10]:
```
Yes      -0.174771
No        0.737300
RepYes    0.731075
RepNo    -0.146779
DemYes   -0.164200
DemNo     1.060967
dtype: float64
```

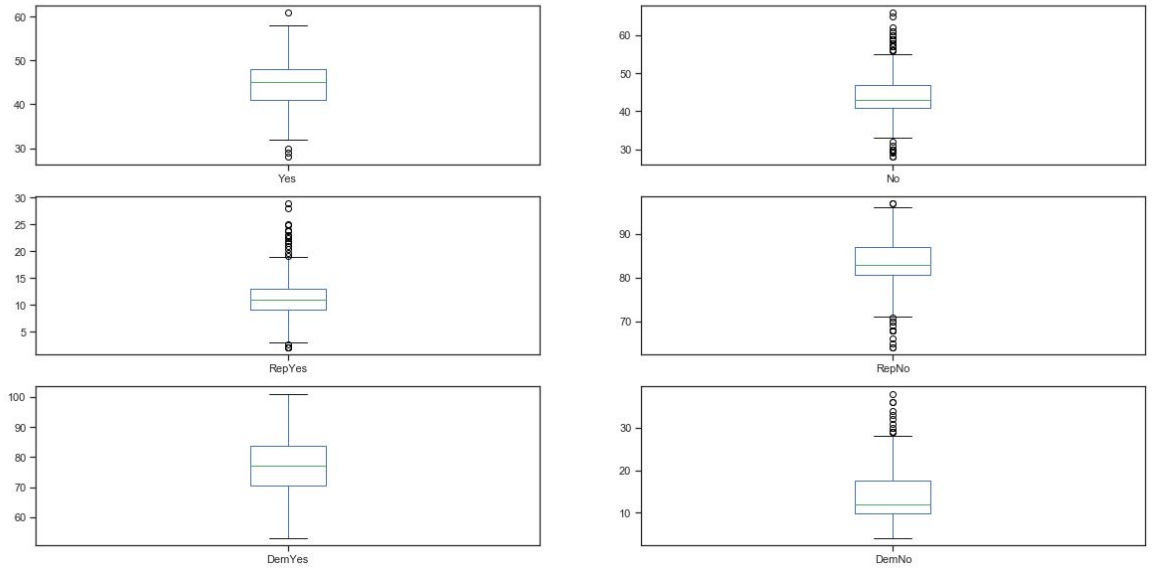
```
[60]: # histogram
hist = data_sub.hist(bins=20, figsize=(20,10))
```



Box Plot

The best tool to identify the outliers is the box plot. Through box plots, we find the minimum, lower quartile (25th percentile), median (50th percentile), upper quartile (75th percentile), and a maximum of an continues variable. Each horizontal line starting from bottom will show the minimum, lower quartile, median, upper quartile and maximum value. We can use a box plot to explore the distribution of a continues variable across the strata.

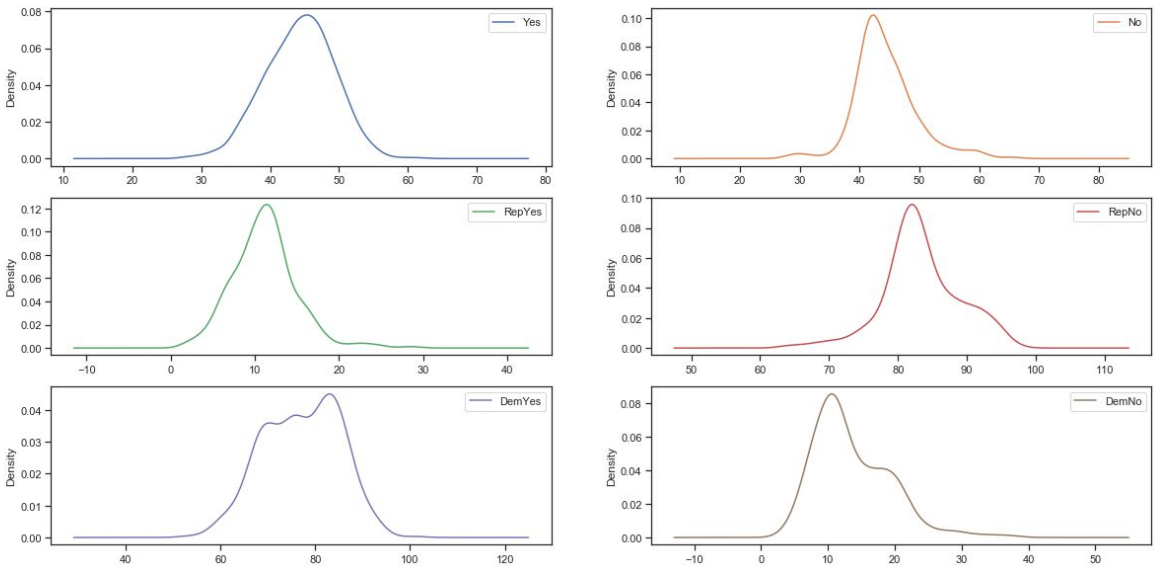
```
[61]: # box plot
box = data_sub.plot(kind='box', subplots=True, layout=(3,2),sharex=False,sharey=False, figsize=(20,10))
box1= data_sub.plot(kind='box', subplots=False, layout=(2,3), sharex=False, sharey=False, figsize=(20,10))
```



Density Plot

Similar to the histogram, the density plots are used to show the distribution of data. Additionally, density plots are especially useful for comparison of distributions. Also, with density plots, we can illustrate how the distribution of a particular variable changes over time. An advantage Density Plots have over Histograms is that they're better at determining the distribution shape because they're not affected by the number of bins used (each bar used in a typical histogram).

```
[62]: # density plot
dens = data_sub.plot(kind='density', subplots=True, layout=(3,2), sharex=False, sharey=False,figsize=(20,10))
```

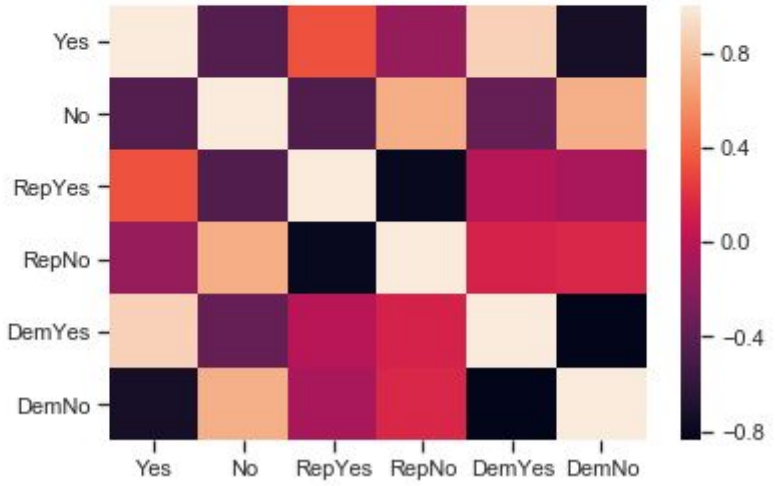


Correlation Plot Matrix

A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables.

```
[14]: # correlation heat map
      corr = data_sub.corr()
      sns.heatmap(corr,
                  xticklabels=corr.columns.values,
                  yticklabels = corr.columns.values)
```

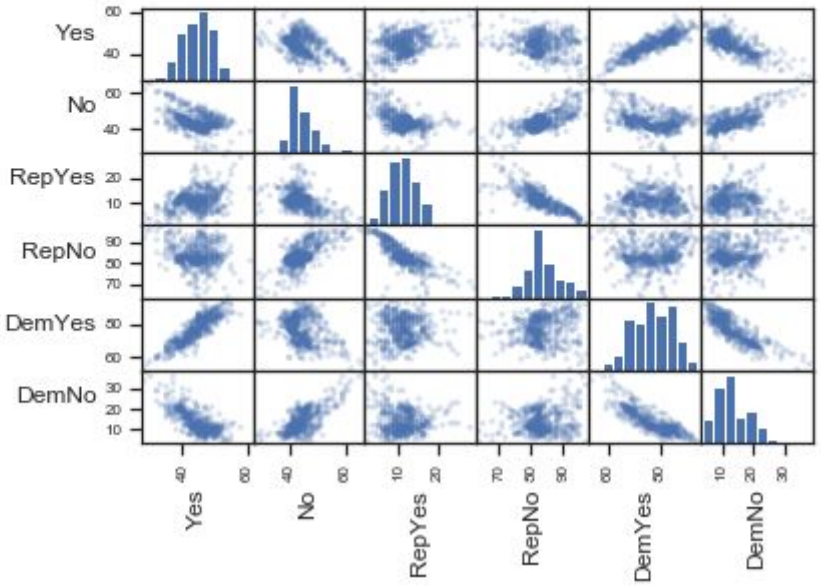
```
[14]: <matplotlib.axes._subplots.AxesSubplot at 0x5e058f0>
```



Scatter Plot Matrix

Each scatter plot in the matrix visualizes the relationship between a pair of variables. Attributes with structured relationships may also be correlated and good candidates for removal from the dataset.

```
[15]: # pandas scatter_matrix
ocor_a = pd.plotting.scatter_matrix(data_sub, alpha = 0.2)
for i in ocor_a.flatten():
    i.xaxis.label.set_rotation(90)
    i.yaxis.label.set_rotation(0)
    i.yaxis.label.set_ha('right')
```



Imputation of 'RepYes' missing values with Linear Regression

The straight line can be seen in the plot, showing how linear regression attempts to draw a straight line that will best minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation.

A variable 'filter' was created to use only non null values as a training dataset.

k-Fold Cross-Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
    a. Take the group as a hold out or test data set
    b. Take the remaining groups as a training data set
    c. Fit a model on the training set and evaluate it on the test set
    d. Retain the evaluation score and discard the model
4. Summarize the skill of the model using the sample of model evaluation scores.

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times.

[63]:
```
# using k-fold validation for the model using 10 folds.
kf = KFold(n_splits=10, random_state = 42)
for train_index, test_index in kf.split(df_filter):
    df_test = df_filter.iloc[test_index]
    df_train = df_filter.iloc[train_index]
# defining input and output.
    x_train = np.array(df_train['Yes']).reshape(-1,1)
    y_train = np.array(df_train['RepYes']).reshape(-1,1)
    x_test = np.array(df_test['Yes']).reshape(-1,1)
    y_test = np.array(df_test['RepYes']).reshape(-1,1)
# fiting LR model.
    model = LinearRegression()
    model.fit(x_train, y_train)
# generating/appending prediction values to the objects created before
    y_pred.append(model.predict(x_test)[0])
    y_true.append(y_test[0])
```

[21]:
```
# creating list with NaN values on RepYes to be used in the model
df_missing = data_sub[data_sub['RepYes'].isnull()].copy()
```

[65]:
```
# predicting NaN values on RepYes with the model for NaN values
x_test_lr = np.array(df_missing['Yes']).reshape(-1,1)
y_test_lr = np.array(df_missing['RepYes']).reshape(-1,1)

x_train_lr = np.array(df_filter['Yes']).reshape(-1,1)
y_train_lr = np.array(df_filter['RepYes']).reshape(-1,1)
```

[24]:
```
# creating LR object
model_lr = LinearRegression()
```

[25]:
```
# fiting LR model
model_lr.fit(x_train_lr, y_train_lr)
print('Linear regression predictions: ', model_lr.predict(x_test_lr)[0])
```

```
Linear regression predictions:  [9.2239262]
```

[27]:
```
# store prediction result in a variable
pred = model_lr.predict(x_test_lr)
```

[29]:
```
# crating variable with RepYes values to imput predicted values in NaN psitions
repyes_vals = data_sub['RepYes'].values
```

[30]:
```
# loop to imput values on RepYes NaN with predicted values
i_value = 0
for i in range(len(repyes_vals)):
    if np.isnan(repyes_vals[i]):
        repyes_vals[i] = pred[i_value]
        i_value += 1
```

[31]:
```
# copiando values from variable to RepYes column
data_sub['RepYes'] = repyes_vals
```

Verifying the model

Mean Squared Errors: is a simple method to fit intercept lines between points and compare those lines to find out the best fit through error reduction. The errors are the sum difference between actual value and predicted value. The mean squared error tells you how close a regression line is to a set of points. It does this by taking the distances from the points to the regression line (these distances are the “errors”) and squaring them. The squaring is necessary to remove any negative signs. It also gives more weight to larger differences. It’s called the mean squared error as you’re finding the average of a set of errors. The smaller the means squared error, the closer you are to finding the line of best fit.

[37]:
```
# checking performance of model with mean square error
print(len(y_pred))
print(len(y_true))
print('Mean Square Error: ', mean_squared_error(y_true, y_pred))
```

```
10
10
Mean Square Error:  15.953709449188173
```

Applying the Model

```
In [41]: print('Linear regression predictions: ', model_lr.predict(x_test_lr)[0])
Linear regression predictions:  [68.10481426]
```

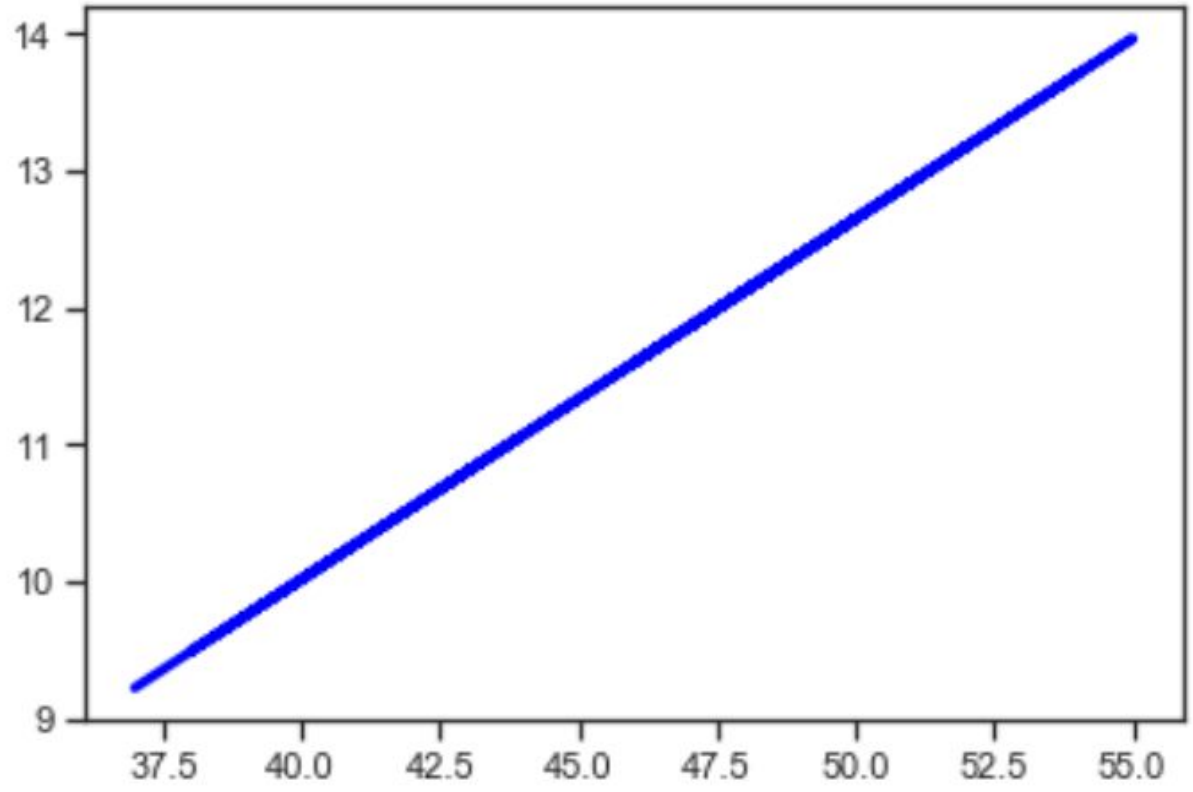
The coefficients

```
In [48]: print('Coefficients: \n', model_lr.coef_)
Coefficients:
 [[0.26323889]]
```

For Mean Squared Error and Coefficient of Determination as well as the Scatter Plot are not possible to be generated for this predicted model as the sample predicted was NaN values so there is no comparison possible.

Plot

The straight line can be seen in the plot, showing how linear regression attempts to draw a straight line that will best minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation.



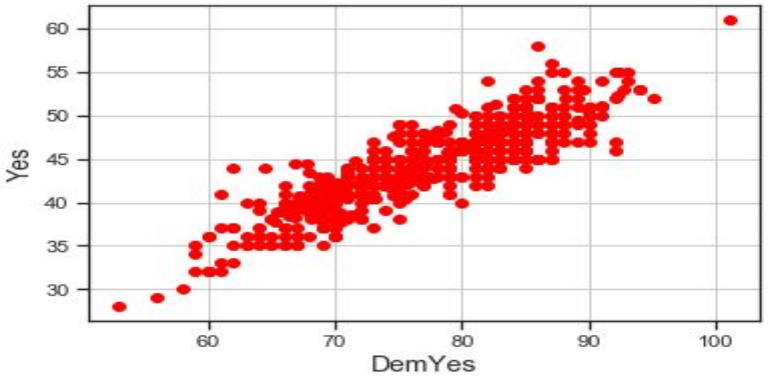
Similar process was developed for ‘DemYes’ predictive imputation of missing values. The code is attached.

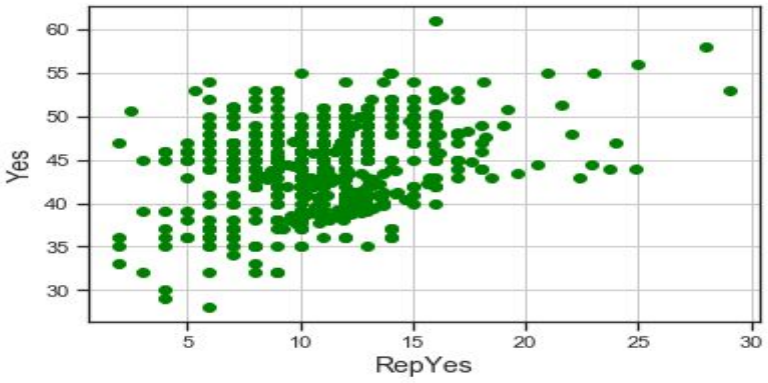
Multilinear Regression

[50]:
```
# Multivariate Linear Regression to predict Yes

# creating different dataframe with columns to be used as target and variables
df = pd.DataFrame(data_sub, columns=['Yes','RepYes','DemYes'])
```

[51]:
```
# creating 'x' and 'y' objects to be used into the Multiple Linear Regression
x = pd.DataFrame(df, columns=['RepYes','DemYes'])
y = pd.DataFrame(df, columns=['Yes'])
```

[52]:
```
# ploting to see linear relationship
plt.scatter(df['DemYes'], df['Yes'], color='red')
plt.xlabel('DemYes', fontsize=14)
plt.ylabel('Yes', fontsize=14)
plt.grid(True)
plt.show()

plt.scatter(df['RepYes'], df['Yes'], color='green')
plt.xlabel('RepYes', fontsize=14)
plt.ylabel('Yes', fontsize=14)
plt.grid(True)
plt.show()
```





[53]:
```
# spliting database
k2 = int(len(df['Yes']) * 0.2) # 20% samples
k8 = int(len(df['Yes']) * -0.8) # 80% samples

x_train = x[k8:] #80%
x_train = np.c_[np.ones(len(x_train),dtype='int64'),x_train]
y_train = y[k8:]

x_test = x[:k2] #20%
x_test = np.c_[np.ones(len(x_test),dtype='int64'),x_test]
y_test = y[:k2]
y_test = y_test.round(0)
```

[54]:
```
# creating LR object
model_mr = LinearRegression()
```

[55]:
```
# fiting LR model
model_mr.fit(x_train, y_train)
print('multi linear regression predictions: ', model_mr.predict(x_test)[0])
```

```
multi linear regression predictions:  [33.78166857]
```

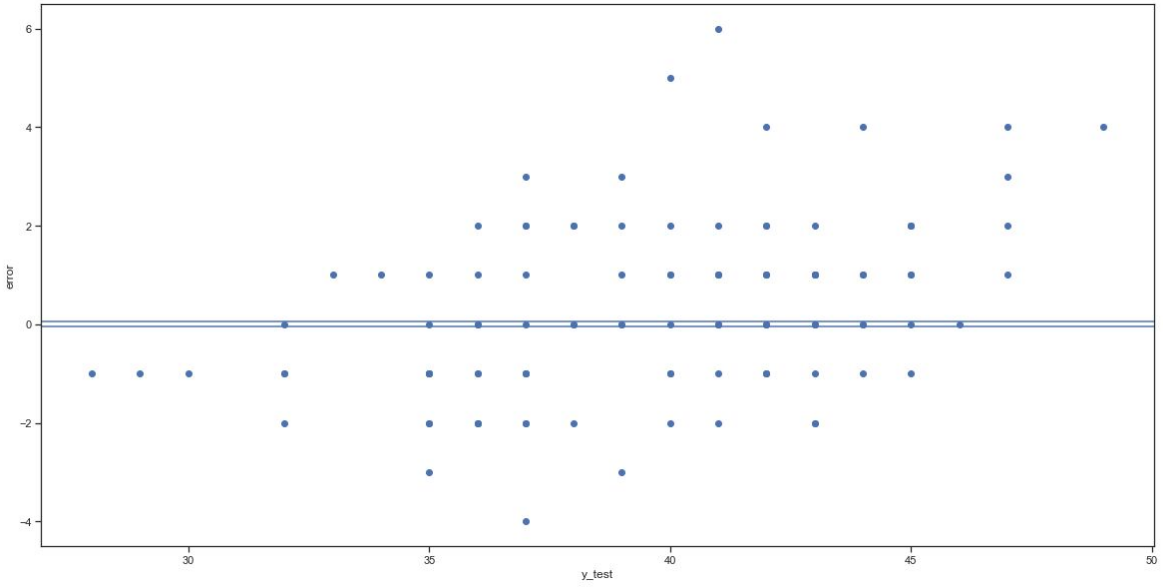
[56]:
```
# store prediction result in a variable
pred = model_mr.predict(x_test)
pred = pred.round(0)
```

[57]:
```
# verifying accuracy
# coefficients
print('Coefficients: \n', model_mr.coef_)
# mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test, pred))
# variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, pred))
```

```
Coefficients:
 [[0.         0.40688594 0.54404073]]
Mean squared error: 3.13
Variance score: 0.82
```

[58]:
```
# prepare model data point for visualization
pred_f = pred.flatten()
pred_df = pd.DataFrame({'Predict_Yes':pred_f})

dfk = pd.concat([y_test, pred_df], axis=1)

error = dfk['Yes'] - dfk['Predict_Yes']
```

[59]:
```
# plot error
plt.figure(figsize = (20,10))
plt.scatter(y_test,error)
plt.axhline(0.05)
plt.axhline(-0.05)
plt.xlabel('y_test')
plt.ylabel('error')
plt.show()
```



Classification Model

Classification can be performed on structured or unstructured data. Classification is a technique where we categorize data into a given number of classes. The main goal of a classification problem is to identify the category/class to which a new data will fall under.

Common terminologies:

- Classifier: An algorithm that maps the input data to a specific category.
- Classification model: A classification model tries to draw some conclusion from the input values given for training. It will predict the class labels/categories for the new data.
- Feature: A feature is an individual measurable property of a phenomenon being observed.
- Binary Classification: Classification task with two possible outcomes. Eg: Gender classification (Male / Female)
- Multi-class classification: Classification with more than two classes. In multi class classification each sample is assigned to one and only one target label. Eg: An animal can be cat or dog but not both at the same time
- Multi-label classification: Classification task where each sample is mapped to a set of target labels (more than one class). Eg: A news article can be about sports, a person, and location at the same time.
-

The following are the steps involved in building a classification model:

- Initialize the classifier to be used.
- Train the classifier: All classifiers in scikit-learn uses a fit(X, y) method to fit the model(training) for the given train data X and train label y.
- Predict the target: Given an unlabeled observation X, the predict(X) returns the predicted label y.
- Evaluate the classifier model

Database

This dataset describes the results of a survey of 1,615 adult men conducted by SurveyMonkey in partnership with FiveThirtyEight and WNYC Studios from May

10-22, 2018. It contains 1615 lines distributed over 65 columns. For this work NaN values will be excluded and 15 columns will be used.

Important highlight that categorical columns will be created for each column as the columns are textual.

The classification goal is to label the feature “q0001” “In general, how masculine or “manly” do you feel?” based on other 14 features with a possible 4 answers:

1. Very masculine
2. Somewhat masculine
3. Not very masculine
4. Not at all masculine

Data Inspection

After loading the data the missing values were excluded as they have no relevance for the work. There will rest 1586 lines and 15 columns.

[2]:
```
# read file
data_set = pd.read_csv('raw-responses-1.csv')
data_set = data_set.dropna()
```

Subsetting the dataset and creating a variable to store the new 15 columns names to be used in the categorical columns which will be created based in the original descriptive columns.

[4]:
```
# subset with necessary columns only
columns_or = ['q0001','q0002','q0005','q0008_0001','q0008_0002','q0008_0003','q0008_0004','q0008_0005'
              ,'q0008_0006','q0008_0007','q0008_0008','q0008_0009','q0008_0010','q0008_0011','q0008_0012'
              ,'q0017','q0018','q0022','q0024','q0026','age3','q0028','q0029' ]

columns_cat = ['q0001_cat','q0002_cat','q0005_cat','q0008_0001_cat','q0008_0002_cat','q0008_0003_cat'
               ,'q0008_0004_cat','q0008_0005_cat','q0008_0006_cat','q0008_0007_cat','q0008_0008_cat'
               ,'q0008_0009_cat','q0008_0010_cat','q0008_0011_cat','q0008_0012_cat','q0017_cat','q0018_cat'
               ,'q0022_cat','q0024_cat','q0026_cat','age3_cat','q0028_cat','q0029_cat']


data_sub = data_set[columns_or]
```

Using a loop to change all columns type to ‘category’ as this is necessary to use the function to automatically create categorical values from a regular column.

[6]:
```
# categorical encoding (text data into numerical)
obj_df = data_sub

# change all columns to 'category' type to apply categorical transformation
for col in [columns_or]:
    obj_df[col] = obj_df[col].astype('category')
```

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution. A Gaussian distribution is also called Normal distribution. Gaussian Naive Bayes is a variant of Naive Bayes that follows Gaussian normal distribution and supports continuous data. Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes’ theorem with the “naive” assumption of conditional independence between every pair of features given the value of the class variable. When working with continuous data, an assumption often taken is that the continuous values associated with each class are distributed according to a normal (or Gaussian) distribution.

:
```
# Gaussian Naive Bayes Classification
```

:
```
# creating 'x' and 'y' objects to be used into the Multiple Linear Regression
x = pd.DataFrame(df_cat, columns=['q0002_cat','q0005_cat','q0008_0001_cat','q0008_0002_cat','q0008_0003_cat'
                ,'q0008_0004_cat','q0008_0005_cat','q0008_0006_cat','q0008_0007_cat','q0008_0008_cat'
                ,'q0008_0009_cat','q0008_0010_cat','q0008_0011_cat','q0008_0012_cat','q0017_cat','q0018_cat'
                ,'q0022_cat','q0024_cat','q0026_cat','age3_cat','q0028_cat','q0029_cat'])

y = pd.DataFrame(df_cat, columns=['q0001_cat'])
```

Feature selection is the process of identifying and selecting a subset of input features that are most relevant to the target variable. Select features according to the k highest scores. SelectKBest selects the top k features that have maximum relevance with the target variable. It takes two parameters as input arguments, "k" and the score function to rate the relevance of every feature with the target variable.

Score_func: Function taking two arrays X and y, and returning a pair of arrays (scores, pvalues) or a single array with scores.

Chi2: Chi-squared stats of non-negative features for classification tasks. This score can be used to select the n_features features with the highest values for the test chi-squared statistic from X, which must contain only non-negative features such as

booleans or frequencies (e.g., term counts in document classification), relative to the classes.

[13]:
```
# feature extraction
np.set_printoptions(suppress=True)

test = SelectKBest(score_func=chi2, k=3)
fit = test.fit(x, y)
```

.set_printoptions: These options determine the way floating point numbers, arrays and other NumPy objects are displayed.

[14]:
```
# summarize scores
np.set_printoptions(precision=3)
print(fit.scores_)
```

```
[43.464 23.407  4.023 11.993 19.044 13.7    1.813  5.18  20.796 21.17
  7.242  3.732  1.19   3.774 10.8   80.027  6.009  3.301  7.886  2.077
  1.92   4.463]
```

Once the scores were available the dataset was recreated using the best scored features.

[15]:
```
# recriating x with better scored features
x = pd.DataFrame(df_cat, columns=['q0008_0001_cat','q0029_cat','q0008_0006_cat','q0022_cat','q0008_0009_cat'
                                  ,'q0026_cat','q0017_cat','q0008_0002_cat','q0008_0004_cat','q0008_0003_cat'
                                  ,'q0008_0007_cat','q0008_0008_cat','q0005_cat','q0002_cat','q0018_cat'])
```

To split the database for test and training, differente from the regression exercise in this was used an direct algebraic calculation to separate 20% and 80% samples based on the total lines of the dataset.

[17]:
```
# spliting database
k2 = int(len(df_cat['q0001_cat']) * 0.2) # 20% samples

x_train = x[k2:] #80%
x_train = np.c_[np.ones(len(x_train),dtype='int64'),x_train]
y_train = y[k2:]

x_test = x[:k2] #20%
x_test = np.c_[np.ones(len(x_test),dtype='int64'),x_test]
y_test = y[:k2]
y_test = y_test.round(0)
```

Gaussian naive Bayes:In this classifier, the assumption is that data from each label is drawn from a simple Gaussian distribution.Gaussian distribution, which is called

the standard Gaussian distribution. Standardization in general is accomplished by subtracting the center of the distribution from a given element in the distribution and dividing the result by the standard deviation of the distribution. The distribution of a standardized Gaussian distribution—that is, a Gaussian distribution that has its elements standardized in this form—has its center at zero and has a variance of unity.

```
[18]: # train the model
      clf = GaussianNB()
      clf.fit(x_train, y_train)
```

The target is to predict the results for feature ‘q0001_cat’ which represents 4 possibilities of answers for the original survey. The result is stored and compared to the original sample to provide accuracy of the model. The focus of precision is positive predictions. It indicates how many positive predictions are true.

Checking the performance it is possible to see a percentual of mach around 16%. While there are no consensus about what is a good accuracy, as it might change depending on the data evaluated, it is clear that this is a very poor result and probably would achieve better rates in a guess contest.

```
[23]: # check performance
      matches = (predicted_m == exprected_m)
      correct = (matches.sum() / float(len(matches)))*100
      print('Coredictions match(%): ',correct.round(2))

      print(metrics.confusion_matrix(exprected_m, predicted_m))
```

```
Coredictions match(%):  16.09
[[  1   0   0   0   0]
 [  5   0   0   3   0]
 [ 19   0   4  10   0]
 [104   0   6  39   8]
 [ 93   0   3  15   7]]
```

A starting point to investigate the low accuracy is about the chosen features regarding the target which seems "noisy" and non-informative to work with a classifier which, besides important for the survey, seems to be not good enough for this work goal. Even the methodology(or lack of it) used to split the sample for test and training might have influenced these results. There are a few approaches this work could try to improve this rate:

1. Standardizing numerical features,

2. Encoding categorical values splitting the data in different columns with zeros and ones.
3. Resampling the unbalanced entries in the minority classes.
4. Stratified K-fold which is also a resource to balance targets of the dataset.
5. Change the classification algorithm and compare the new results with the original Gaussian Naive Bayes.

REFERENCES

Bycoffe, A. Mehta, D. Silver, N. (2020) ‘How unpopular is Donald Trump?’, FiveThirtyEight. Available at https://projects.fivethirtyeight.com/trump-approval-ratings/ (Accessed: 4 Nov 2020).

Rouse, M. (2018) ‘data sampling’, SearchBusinessAnalytics. Available at https://searchbusinessanalytics.techtarget.com/definition/data-sampling#:~:text=Data%20sampling%20is%20a%20statistical,larger%20data%20set%20being%20examined. (Accessed: 29 Oct 2020).

Kaustubh N. (2017) ‘Chinese Restaurant Process’, Medium. Available at https://medium.com/somx-labs/chinese-restaurant-process-1bf036b5bf0f (Accessed: 29 Oct 2020).

Subramanian, D. (2019) ‘Decision Tree in Layman’s Terms’, Towards Data Science. Available at https://towardsdatascience.com/decision-tree-in-laymans-terms-part-1-76e1f1a6b672 (Accessed: 28 Oct 2020).

Chauhan, N. (2020) ‘Decision Tree Algorithm, Explained’, kdnuggets. Available at https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html (Accessed: 28 Oct 2020).

Kassambara, A. (2019) ‘Hierarchical Clustering In R: The Essentials’, Data Novia. Available at https://www.datanovia.com/en/lessons/agglomerative-hierarchical-clustering/ (Accessed: 28 Oct 2020).

Patlolla, C. (2018) ‘Understanding the concept of Hierarchical clustering Technique’, Towards Data Science. Avaliable at

https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec (Accessed: 29 Oct 2020).

sklearn.feature_selection.SelectKBest, Scikit-Learn. Available at https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html (Accessed: 15 Nov 2020)

Pedregosa et al. (2011) 'Scikit-learn: Machine Learning in Python', JMLR 12, pp. 2825-2830.

Pierre, S. (2019) 'Predicting Missing Values with Python', Towards Data Science. Avaliable at https://towardsdatascience.com/predicting-missing-values-with-python-41e581511117 (Accessed: 15 Nov 2020).

Statistics Solutions, (2013). 'What is Linear Regression'. Avaliable at https://www.statisticssolutions.com/what-is-linear-regression/ (Accessed 20 Nov 2020)

Yale University, (1998). 'Correlation'. Available at http://www.stat.yale.edu/Courses/1997-98/101/correl.htm (Accessed 20 Nov 2020)

Bouganim, L. (2009). 'Data Skew', Encyclopedia of Database Systems. Available at https://doi.org/10.1007/978-0-387-39940-9_1088 (Accessed 20 Nov 2020)

Dhana, K. (2017). 'Use Box Plots to Assess the Distribution and to Identify the Outliers in Your Dataset', datascience+. Available at https://datascienceplus.com/box-plots-identify-outliers/ (Accessed 20 Nov 2020)

Brownlee, J. (2018), 'A Gentle Introduction to k-fold Cross-Validation', Machine Learning Mastery. Available at https://machinelearningmastery.com/k-fold-cross-validation/ (Accessed 20 Nov 2020)

Glen, S. (2013). ‘Mean Squared Error: Definition and Example’, StatisticsHowTo.com. Available at https://www.statisticshowto.com/mean-squared-error/#:~:text=The%20mean%20squared%20error%20tells,errors%E2%80%9D)%20and%20squaring%20them.&text=It's%20called%20the%20mean%20squared,of%20a%20set%20of%20errors. (Accessed 20 Nov 2020)

Garg, R. (2018). ‘7 Types of Classification Algorithms’, Analytics India Magazine chronicles. Available at https://analyticsindiamag.com/7-types-classification-algorithms/#:~:text=Classification%20model%3A%20A%20classification%20model,of%20a%20phenomenon%20being%20observed. (Accessed 20 Nov 2020)

‘numpy.set_printoptions’, NumPy.org. Available at https://numpy.org/doc/stable/reference/generated/numpy.set_printoptions.html#:~:text=set_printoptions,-numpy.&text=These%20options%20determine%20the%20way,point%20output%20(default%208). (Accessed 20 Nov 2020)

‘Gaussian Distribution’, ScienceDirect. Available at https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/gaussian-distribution . (Accessed 20 Nov 2020)