Linda Cheluget
13009028
Assignment 2A: Python and R cheatsheet.

A Simple Python and R cheat sheet.

Python R

Data Importation - There are several forms of data that can be imported into Python and R. Python has Pandas library that's built on numpy, provides tools used for data structuring and data analysis. Pandas enables data to be read and written. While R has libraries such as readr that allows one to import csv files.

```python
#Read and Write to CSV
import pandas as pd
data=pd.read_csv('data.csv')
data.head()

#Read and Write to Excel
pd.read_excel('sample.xlsx')
```

```r
#Read and Write to CSV
library(readr)
dataset <- read.csv("data.csv")
head(dataset)

#Read and Write to Excel
```

Variables with a unique value (unique_count = 1) is usually excluded from data analysis. If the type of data used is not an integer or numeric and unique_rate = 1 then the variable will most likely be an identifier.

```python
#checking uniqueness
data.age.unique()
#unique entries
data.age.unique().shape
```

```r
#checking uniqueness
unique(dataset$age)
#unique entries
length(unique(dataset$age))
```

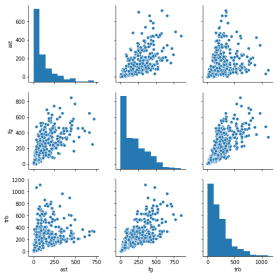
The function below is equivalent to as.data.frame(table(x)) Arguments can be from df, vars and wt_var.

```python
#counting
data.player.value_counts()
```

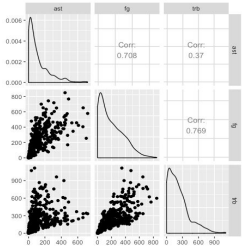
```r
#counting
sort(table(dataset$player),
decreasing = TRUE)
```

Python - seaborn is used for statistical data visualization. This data visualisation library is based on matplolib. Provides interface for creating statistical graphics.
R - Uses Pipes (%>%) as a way of making data manipulation better, to visualise effectively dplyr, pipes %>%, purrr,ggally and ggpairs is combined. Purr applies functions to data ggally which is extended from ggplot2 reduces complexity and Ggpairs pairs plot.

```python
#visualising data

import seaborn as sns
import matplotlib.pyplot as plt
sns.pairplot(data[["ast", "fg",
"trb"]])
plt.show()
```

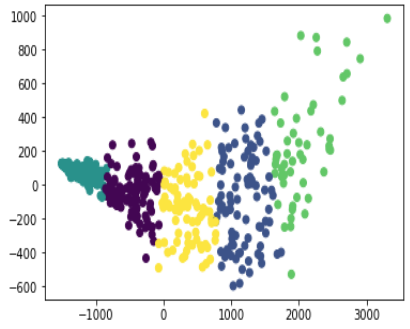


```r
#visualising data

library(purrr)
library(dplyr)
dataset %>%
  select_if(is.numeric) %>%
  map_dbl(mean, na.rm = TRUE)
library(GGally)
dataset %>%
  select(ast, fg, trb) %>%
  ggpairs()
```

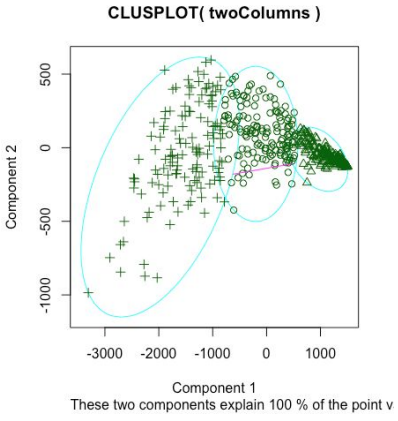


K-means clustering is used on unsupervised data, it partitions it into a set of k groups. k represents the number of groups specified.In k-means clustering, each cluster is represented by its center which is the mean of points assigned to the cluster.

```
#cluster kmeans

from sklearn.cluster import KMeans
kmeans_model = KMeans(n_clusters=5,
random_state=1)
columns =
data._get_numeric_data().dropna(axi
s=1)
kmeans_model.fit(columns)
labels = kmeans_model.labels_

from sklearn.decomposition import
PCA
pca_2 = PCA(2)
plot_columns =
pca_2.fit_transform(columns)
plt.scatter(x=plot_columns[:,0],
y=plot_columns[:,1], c=labels)
plt.show()
```




```
#cluster kmeans

library(cluster)
set.seed(1)
column <- function(col){
  sum(is.na(col)) == 0 &&
is.numeric(col)
}
newcolumn <- sapply(dataset,
column)
clusters <-
kmeans(dataset[,newcolumn],
centers=3)
labels <- clusters$cluster

dataset1 <-
prcomp(dataset[,newcolumn],
center=TRUE)
twocolumns <- dataset1$x[,1:2]
clusplot(twoColumns, labels)
```



Train/Test Split.The data used to create models is normally split into train and test data, the training set has a known output that the model will learn on, the subset which is a test dataset is used to test the model's prediction.

```
#split dataset train/test

from sklearn.model_selection import
train_test_split
x_train, x_test, y_train, y_test =
train_test_split(data[['fg']],
data[['ast']], test_size=0.2,
random_state=42)
```

```
#split dataset train/test

sample_size <- floor(0.8 *
nrow(dataset))
set.seed(1)
trainIndex <-
sample(1:nrow(dataset),
sample_size)
train <- dataset[trainIndex,]
test <- dataset[-trainIndex,]
```

Linear regression tries to model the relationship between two variables by using a linear equation through fitting, one variable is explanatory while the other is dependent.

```
#creating models
#creating a Linear Regression Model

from sklearn.linear_model import
LinearRegression
model = LinearRegression()
model.fit(x_train, y_train) #Train
the model
model.score(x_train, y_train)
predictions = model.predict(x_test)
#Make predictions on the test data
score = model.score(x_train,
y_train)
print(score)

#0.4840071637351546
```

```
#creating models
#creating a Linear Regression Model

linear <- lm(ast ~ fg, data=train)
predictions <- predict(linear,
test)
summary(linear)

#0.505
```

Logistic regression is just like any other regression analysis, but it specifically focuses predictive analysis.Logistic regression shows the relationship between one dependent variable(binary) and one or more nominal variable.

```
#creating a Logistic Regression

from sklearn.linear_model import
LogisticRegression
model2 = LogisticRegression()
model2.fit(x_train, y_train) #Train
the model
predictions =
model2.predict(x_test) #Make
predictions on the test data
model2.score(x_train, y_train)
score = model2.score(x_train,
y_train)
print(score)

#0.6979166666666666
```

```
#creating a logistic regression

logistic <- glm(ast ~ fg,
data=train)
predicted= predict(logistic,test)
summary(logistic)

# 0.699
```

Decision tree learning is a method commonly used in data mining. Its aim is to produce a model that uses target variable value based on multiple variable input.

```
#creating a decision tree

from sklearn.tree import
DecisionTreeClassifier
model3 = DecisionTreeClassifier()
#model3 =
DecisionTreeClassifier(criterion =
"entropy", splitter = "random",
max_depth = 2,  min_samples_split =
5, min_samples_leaf = 2,
max_features = 2)
model3.fit(x_train, y_train)
model3.score(x_train, y_train)
predictions =
model3.predict(x_test) #Make
predictions on the test data
score = model3.score(x_train,
y_train)
```

```
#creating a decision tree

library(rpart)
#tree
fit <- rpart(ast ~ fg, data=train,
method="class")
predicted= predict(fit,test)
summary(fit)
```

Gradient boosting. Gradient boosting is a machine learning technique that provide solutions for classification and regression issues. The prediction model is produced in the form of a collection of weak prediction models like decision trees.

```
#creating a gradient boost

from sklearn.ensemble import
GradientBoostingClassifier
model4=
GradientBoostingClassifier(n_estima
tors=100, \learning_rate=1.0,
max_depth=1, random_state=0)
model4.fit(x_train, y_train)
model4.score(x_train, y_train)
predictions =
model4.predict(x_test) #Make
predictions on the test data
score = model4.score(x_train,
y_train)
print(score)
```

```
#creating a gradient boost

library(caret)
fit <- train(ast ~ fg, data=train,
method = "gbm")
predicted= predict(fit,test,type=
"prob")[,2]
summary(fit)
```