```python
# -*- coding: utf-8 -*-
"""
Created on Tue Jun  1 09:47:01 2021

@author: Sarah
"""

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import sklearn

# Split data ready for ML
dataset = pd.read_csv('Dataset_Final_Numpy.csv')
print(dataset.head)

dataset.columns = ['T_xacc', 'T-yacc', 'T_zacc', 'T_xgyro',
       'T_ygro', 'T_zgyro', 'RA_xacc', 'RA_yacc', 'RA_zacc', 'RA_xgyro',
       'RA_ygro', 'RA_zgyro', 'LA_xacc','LA_yacc', 'LA_zacc', 'LA_xgyro',
       'LA_ygyro', 'LA_zgryo', 'RL_xacc', 'RL_yacc', 'RL_zacc', 'RL_xgyro',
       'RL_ygyro', 'RL_zgyro', 'LL_xacc','LL_yacc', 'LL_zacc', 'LL_xgyro',
       'LL_ygyro', 'LL_zgyro', 'Subject', 'Activity']
print(dataset.head)
X = np.array(dataset.drop(['Activity', 'Subject'],1))
print(X)
np.where(np.isnan(X))
y = np.array(dataset['Activity'])
print(X.shape)
print(y.shape)

# Do we need to scale? (see below)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35,
                                                    random_state=0)
print(X_train.shape)
print(X_train)
print(X_test.shape)
print(X_test)
print(y_train.shape)
print(y_train)
print(y_test.shape)
print(y_test)

# KNeighboursClassifier - 3
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train, y_train)
model = clf.predict(X_test)
print(model)
print("Test set accuracy: {:.2f}".format(clf.score(X_test, y_test)))

# KNeighboursClassifier - 5
clf = KNeighborsClassifier(n_neighbors=5)
clf.fit(X_train, y_train)
model = clf.predict(X_test)
print(model)
```

```python
print("Test set accuracy: {:.2f}".format(clf.score(X_test, y_test)))

# Repeat with smaller training set (55% of data)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.45,
                                                    random_state=0)
# KNeighboursClassifier - 3
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train, y_train)
model = clf.predict(X_test)
print(model)
print("Test set accuracy: {:.2f}".format(clf.score(X_test, y_test)))

# KNeighboursClassifier - 5
clf = KNeighborsClassifier(n_neighbors=5)
clf.fit(X_train, y_train)
model = clf.predict(X_test)
print(model)
print("Test set accuracy: {:.2f}".format(clf.score(X_test, y_test)))

# Repeat with larger training set (85% of data)

# Logistic Regression
from sklearn.linear_model import LogisticRegression
# Struggle with optimisation of models: number of iterations
# Scoring - default is accuracy
# For multiclass problems, only ‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’ handle multinomial loss

# With sag solver: 64.9%
clf = LogisticRegression(C=1, multi_class='multinomial', fit_intercept=True, penalty='l2',
                   tol=0.0001, max_iter=10000, solver='sag')
clf.fit(X,y)
print(np.exp(clf.intercept_),np.exp(clf.coef_.ravel()))
print(clf.score(X, y))

# With newton-cg solver: 65.0%
clf = LogisticRegression(C=1, multi_class='multinomial', fit_intercept=True, penalty='l2',
                   tol=0.0001, max_iter=10000, solver='newton-cg')
clf.fit(X,y)
print(np.exp(clf.intercept_),np.exp(clf.coef_.ravel()))
print(clf.score(X, y))

# With saga and L1 penalty: 64.9%
clf = LogisticRegression(C=1, multi_class='multinomial', fit_intercept=True, penalty='l1',
                   tol=0.0001, max_iter=10000, solver='saga')
clf.fit(X,y)
print(np.exp(clf.intercept_),np.exp(clf.coef_.ravel()))
print(clf.score(X, y))

# With lbfgs and L2 penalty: 65.0%
clf = LogisticRegression(C=1, multi_class='multinomial', fit_intercept=True, penalty='l2',
                   tol=0.0001, max_iter=10000, solver='lbfgs')
clf.fit(X,y)
print(np.exp(clf.intercept_),np.exp(clf.coef_.ravel()))
print(clf.score(X, y))

# Linear Discriminant Analysis: 60.1%
```

```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
model = LinearDiscriminantAnalysis(solver='svd', shrinkage=None, priors=None,
        n_components=None, store_covariance=False, tol=0.0001)
model.fit(X,y)
print(model.predict(X))
print(model.score(X,y))

# Bayes Theorem for Classification: 77.0% on 50% train / 77.3% on 65% train
# MultinomialNB can't process negative values
from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)
model.fit(X,y)

from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
model = GaussianNB(priors=None, var_smoothing=1e-09)
y_pred = model.fit(X_train, y_train).predict(X_test)
print("Number of mislabeled points out of a total %d points : %d" % (X_test.shape[0],
                                                    (y_test != y_pred).sum()))

print(model.score(X,y))

# Change training data size to 65%: improved performance
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=0)
model = GaussianNB(priors=None, var_smoothing=1e-09)
y_pred = model.fit(X_train, y_train).predict(X_test)
print("Number of mislabeled points out of a total %d points : %d" % (X_test.shape[0],
                                                    (y_test != y_pred).sum()))

print(model.score(X,y))

# Support Vector Machines
from sklearn import svm
clf = svm.LinearSVC(penalty='l2', loss='squared_hinge', dual=True, tol=0.0001,
    C=1, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None,
    verbose=0, random_state=None, max_iter=200000)
clf.fit(X, y)
print(clf.score(X,y))
# Converged on 200,000 with 62% accuracy score

# Still 62% accuracy with L1
from sklearn import svm
clf = svm.LinearSVC(penalty='l1', loss='squared_hinge', dual=False, tol=0.0001,
    C=1, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None,
    verbose=0, random_state=None, max_iter=200000)
clf.fit(X, y)
print(clf.score(X,y))

# Increasing C:
# https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-ke
# 100, 1000, or 10000 stays at 65.0%
```

APPENDIX B: DATA EXPLORATION: R STUDIO

```
library(tidyverse)
body_movement = read_csv('Dataset_Final.csv')
as_tibble(body_movement)

body_movement_sub = select(body_movement, -Subject, -Activity)
summary(body_movement_sub)

# gyro = clockwise negative / anti-clockwise positive. Need to measure amount
of movement
# acceleration & deceleration not continual movement
# Three axes: x,y,z

ggplot(body_movement) +
  aes(x = "", y = T_xacc) +
  geom_boxplot(fill = "#0c4c8a")

ggplot(body_movement) +
  aes(x = "", y = T_yacc ) +
  geom_boxplot(fill = "#0c4c8a")

ggplot(body_movement) +
  aes(x = "", y = T_zacc) +
  geom_boxplot(fill = "#0c4c8a")

library(dplyr)
library(purrr)
library(ggplot2)
mean_data = map_dbl(body_movement_sub, mean)
mean_body_movement = as_tibble(mean_data)
mean_body_movement

ggplot(mean_body_movement) +
  geom_jitter(aes(x="", y=mean_data)) +
  ggtitle("Mean of Body Movement Measurements for each Body Part Sensor") +
  ylab("Mean of Sensor Measurements for Acceleration & Rotation")
# Mean looks unlikely to help at this stage
# Can we confirm Gaussian distribution?

variance_data = map_dbl(body_movement_sub, var)
variance_body_movement = as_tibble(variance_data)
variance_body_movement
ggplot(variance_body_movement) +
  geom_jitter(aes(x="", y=variance_data)) +
  ggtitle("Variance of Body Movement Measurements for each Body Part Sensor")
+
  ylab("Variance of Sensor Measurements for Acceleration & Rotation")

Avg_movement_running_subject_1 = rowMeans(body_movement[10000:10249,c(1,16)])
Avg_movement_running_subject_2 = rowMeans(body_movement[10250:10499,c(1,16)])
Avg_movement_running_subject_3 = rowMeans(body_movement[10500:10749,c(1,16)])
Avg_movement_running_subject_4 = rowMeans(body_movement[10750:10999,c(1,16)])
Avg_movement_running_subject_5 = rowMeans(body_movement[11000:11249,c(1,16)])
Avg_movement_running_subject_6 = rowMeans(body_movement[11250:11499,c(1,16)])
Avg_movement_running_subject_7 = rowMeans(body_movement[11500:11749,c(1,16)])
```

```
Avg_movement_running_subject_8 = rowMeans(body_movement[11750:11999,c(1,16)])
boxplot(Avg_movement_running_subject_1, Avg_movement_running_subject_2,
Avg_movement_running_subject_3,
        Avg_movement_running_subject_4, Avg_movement_running_subject_5,
Avg_movement_running_subject_6,
        Avg_movement_running_subject_7, Avg_movement_running_subject_8,
        main = "Average Body Movements Whilst Running for Subjects 1-8",
        ylab = "Average Body Movements (Acceleration & Rotation)",
        xlab = "Subjects 1-8")

Avg_movement_sitting = rowMeans(body_movement[1:1999,c(1,16)])
Avg_movement_standing = rowMeans(body_movement[2000:3999,c(1,16)])
Avg_movement_AscendingStairs = rowMeans(body_movement[4000:5999,c(1,16)])
Avg_movement_DescendingStairs = rowMeans(body_movement[6000:7999,c(1,16)])
Avg_movement_TreadmillFlat = rowMeans(body_movement[8000:9999,c(1,16)])
Avg_movement_TreadmillIncline = rowMeans(body_movement[10000:11999,c(1,16)])
Avg_movement_TreadmillRunning = rowMeans(body_movement[12000:13999,c(1,16)])
Avg_movement_Stepper = rowMeans(body_movement[14000:15999,c(1,16)])
Avg_movement_CrossTrainer = rowMeans(body_movement[16000:17999,c(1,16)])
Avg_movement_BikeHorizontal = rowMeans(body_movement[18000:19999,c(1,16)])
Avg_movement_BikeVertical = rowMeans(body_movement[20000:21999,c(1,16)])
boxplot(Avg_movement_sitting, Avg_movement_standing,
Avg_movement_AscendingStairs,
        Avg_movement_DescendingStairs, Avg_movement_TreadmillFlat,
Avg_movement_TreadmillIncline,
        Avg_movement_TreadmillRunning, Avg_movement_Stepper,
Avg_movement_CrossTrainer,
        Avg_movement_BikeHorizontal, Avg_movement_BikeVertical,
        main = "Average Body Movements for 11 Activities",
        ylab = "Average Body Movements (Acceleration & Rotation)",
        xlab = "Activity Classes 0-10")

# Select sensor measurements with greatest variance until all body parts are
covered
body_movement_new = select(body_movement, RL_yacc, LL_yacc, LL_xacc, RL_xacc,
                                RA_xacc, LA_xacc, T_xacc, Activity)

# Do I have the right data to answer this question?

library(GGally)
ggpairs(body_movement_new)

library(olsrr)
model = lm(Activity~RL_yacc+LL_yacc+LL_xacc+RL_xacc+RA_xacc+LA_xacc+T_xacc,
data = body_movement_new)
ols_step_best_subset(model)
```

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Jun  9 13:39:00 2021

@author: Sarah
"""
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import sklearn

# Step 1 - Split Data: Training 65% Test 35%
dataset = pd.read_csv('Dataset_Final_Numpy.csv')
print(dataset.head)

dataset.columns = ['T_xacc', 'T-yacc', 'T_zacc', 'T_xgyro',
       'T_ygro', 'T_zgyro', 'RA_xacc', 'RA_yacc', 'RA_zacc', 'RA_xgyro',
       'RA_ygro', 'RA_zgyro', 'LA_xacc','LA_yacc', 'LA_zacc', 'LA_xgyro',
       'LA_ygyro', 'LA_zgryo', 'RL_xacc', 'RL_yacc', 'RL_zacc', 'RL_xgyro',
       'RL_ygyro', 'RL_zgyro', 'LL_xacc','LL_yacc', 'LL_zacc', 'LL_xgyro',
       'LL_ygyro', 'LL_zgyro', 'Subject', 'Activity']
print(dataset.head)
X = np.array(dataset.drop(['Activity', 'Subject'],1))
print(X)
np.where(np.isnan(X))
y = np.array(dataset['Activity'])
print(X.shape)
print(y.shape)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35,
                                                    random_state=0)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

# Distribution of attributes
print(np.mean(X[:,1]))
print(np.std(X[:,1]))
mean = -0.32
standard_deviation = 2
import scipy.stats
x_values = np.array(X[:,1])
y_values = scipy.stats.norm(mean, standard_deviation)
plt.plot(x_values, y_values.pdf(x_values))
plt.ylabel('Frequency')
plt.xlabel('Sensor Measurement')
plt.title('Distribution of Torso x Accelerator')

# Step 2 - Scale Data
from sklearn import preprocessing
# Scale X_train
scaler = preprocessing.RobustScaler().fit(X_train)
print(scaler)
X_train_scaled = scaler.transform(X_train)
print(X_train_scaled)
```

```
# Scale X_test
scaler = preprocessing.RobustScaler().fit(X_test)
print(scaler)
X_test_scaled = scaler.transform(X_test)
print(X_test_scaled)
# y is multi-class label so doesn't need scaling

# PCA
from sklearn.decomposition import PCA
pca = PCA()
X_train_scaled = pca.fit_transform(X_train_scaled)
X_test_scaled = pca.transform(X_test_scaled)
explained_variance = pca.explained_variance_ratio_
print(explained_variance)
print(sum(explained_variance))
# Screeplot. What does it mean?
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.grid()
plt.show()

pca = PCA(n_components=25)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# KNeighbours after PCA 95%
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train_pca, y_train)
model = clf.predict(X_test_pca)
print(model)
print("Test set accuracy: {:.2f}".format(clf.score(X_test_pca, y_test)))
# Confusion Matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, model, labels=None, sample_weight=None, normalize=None)
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(clf, X_test_pca, y_test)
plt.show()

from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train_scaled, y_train)
model = clf.predict(X_test_scaled)
print(model)
print("Test set accuracy: {:.2f}".format(clf.score(X_test_scaled, y_test)))

# Precision = TP / TP & FP
# Precision 94.9%
from sklearn.metrics import precision_score
print(precision_score(y_test, model, labels=None, pos_label=1, average='micro',
                      sample_weight=None, zero_division='warn'))
# Micro - calculate metrics globally counting total TP, FN, FP

# Recall = TP / TP & FN
# Recall 94.9%
```

```python
from sklearn.metrics import recall_score
print(recall_score(y_test, model, labels=None, pos_label=1, average='micro',
                   sample_weight=None, zero_division='warn'))
# F1 Average 94.9%
from sklearn.metrics import f1_score
print(f1_score(y_test, model, labels=None, pos_label=1, average='micro',
               sample_weight=None, zero_division='warn'))
# F1 All classes
print(f1_score(y_test, model, labels=None, pos_label=1, average=None,
               sample_weight=None, zero_division='warn'))

# Logistic Regression after PCA
# See notes in ML coursework about selection choices
# With newton-cg: 60.4% (PCA) 65%
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(C=1, multi_class='multinomial', fit_intercept=True, penalty='l2',
                   tol=0.0001, max_iter=10000, solver='newton-cg')
clf.fit(X_train_pca,y_train)
print(clf.score(X_test_pca, y_test))
y_pred = clf.predict(X_test_pca)
confusion_matrix(y_test, y_pred, labels=None, sample_weight=None, normalize=None)
plot_confusion_matrix(clf, X_test_pca, y_test)
plt.show()

# F1 Score 60.5%
print(f1_score(y_test, y_pred, labels=None, pos_label=1, average='micro',
               sample_weight=None, zero_division='warn'))
# F1 All classes
print(f1_score(y_test, y_pred, labels=None, pos_label=1, average=None,
               sample_weight=None, zero_division='warn'))

# With saga and L1 penalty: 60.4% (PCA) 65.1%
clf = LogisticRegression(C=1, multi_class='multinomial', fit_intercept=True, penalty='l1',
                   tol=0.0001, max_iter=10000, solver='saga')
clf.fit(X_train_pca,y_train)
print(clf.score(X_test_pca, y_test))
y_pred = clf.predict(X_test_pca)
confusion_matrix(y_test, y_pred, labels=None, sample_weight=None, normalize=None)
plot_confusion_matrix(clf, X_test_pca, y_test)
plt.show()

# With lbfgs and L2 penalty: 65.0%
clf = LogisticRegression(C=1, multi_class='multinomial', fit_intercept=True, penalty='l2',
                   tol=0.0001, max_iter=10000, solver='lbfgs')
clf.fit(X_train_scaled,y_train)
print(clf.score(X_test_scaled, y_test))

# Linear Discriminant Analysis: 59.9%
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
model = LinearDiscriminantAnalysis(solver='svd', shrinkage=None, priors=None,
        n_components=None, store_covariance=False, tol=0.0001)
model.fit(X_train_scaled,y_train)
y_pred = model.predict(X_test_scaled)
print(model.score(X_test_scaled,y_test))
confusion_matrix(y_test, y_pred, labels=None, sample_weight=None, normalize=None)
plot_confusion_matrix(model, X_test_scaled, y_test)
plt.show()
```

```
# With PCA 56%
model = LinearDiscriminantAnalysis(solver='svd', shrinkage=None, priors=None,
        n_components=None, store_covariance=False, tol=0.0001)
model.fit(X_train_pca,y_train)
y_pred = model.predict(X_test_pca)
print(model.score(X_test_pca,y_test))
confusion_matrix(y_test, y_pred, labels=None, sample_weight=None, normalize=None)
plot_confusion_matrix(model, X_test_pca, y_test)
plt.show()

# F1 Score 56.0%
print(f1_score(y_test, y_pred, labels=None, pos_label=1, average='micro',
               sample_weight=None, zero_division='warn'))
# F1 All classes
print(f1_score(y_test, y_pred, labels=None, pos_label=1, average=None,
               sample_weight=None, zero_division='warn'))

# Quadratic Discriminant Analysis (QDA): Each class uses its own estimate of variance (or covarianc
# Flexible Discriminant Analysis (FDA): Where non-linear combinations of inputs is used such as spl
# Regularized Discriminant Analysis (RDA): Introduces regularization into the estimate of the vari

# Bayes Theorem for Classification 76%
from sklearn.naive_bayes import GaussianNB
model = GaussianNB(priors=None, var_smoothing=1e-09)
y_pred = model.fit(X_train_scaled, y_train).predict(X_test_scaled)
print("Number of mislabeled points out of a total %d points : %d" % (X_test.shape[0],
                                                    (y_test != y_pred).sum()))

print(model.score(X_test_scaled,y_test))
confusion_matrix(y_test, y_pred, labels=None, sample_weight=None, normalize=None)
plot_confusion_matrix(model, X_test_scaled, y_test)
plt.show()
# With PCA 65.4%
model = GaussianNB(priors=None, var_smoothing=1e-09)
y_pred = model.fit(X_train_pca, y_train).predict(X_test_pca)
print("Number of mislabeled points out of a total %d points : %d" % (X_test.shape[0],
                                                    (y_test != y_pred).sum()))

print(model.score(X_test_pca,y_test))
confusion_matrix(y_test, y_pred, labels=None, sample_weight=None, normalize=None)
plot_confusion_matrix(model, X_test_pca, y_test)
plt.show()
# Precision 65.4%
print(precision_score(y_test, y_pred, labels=None, pos_label=1, average='micro',
                      sample_weight=None, zero_division='warn'))
# Recall 65.4%
print(recall_score(y_test, y_pred, labels=None, pos_label=1, average='micro',
                   sample_weight=None, zero_division='warn'))
# F1 Score 65.4%
print(f1_score(y_test, y_pred, labels=None, pos_label=1, average='micro',
               sample_weight=None, zero_division='warn'))
# F1 All classes
print(f1_score(y_test, y_pred, labels=None, pos_label=1, average=None,
               sample_weight=None, zero_division='warn'))

# Support Vector Machines 61.3% (Ridge)
```

```python
from sklearn import svm
clf = svm.LinearSVC(penalty='l2', loss='squared_hinge', dual=True, tol=0.0001,
    C=1, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None,
    verbose=0, random_state=None, max_iter=200000)
clf.fit(X_train_scaled, y_train)
y_pred = clf.predict(X_test_scaled)
print(clf.score(X_test_scaled, y_test))
confusion_matrix(y_test, y_pred, labels=None, sample_weight=None, normalize=None)
plot_confusion_matrix(clf, X_test_scaled, y_test)
plt.show()

# With PCA 56.9%
clf = svm.LinearSVC(penalty='l2', loss='squared_hinge', dual=True, tol=0.0001,
    C=1, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None,
    verbose=0, random_state=None, max_iter=200000)
clf.fit(X_train_pca, y_train)
y_pred = clf.predict(X_test_pca)
print(clf.score(X_test_pca, y_test))
confusion_matrix(y_test, y_pred, labels=None, sample_weight=None, normalize=None)
plot_confusion_matrix(clf, X_test_pca, y_test)
plt.show()
# Precision 56.9%
print(precision_score(y_test, y_pred, labels=None, pos_label=1, average='micro',
                      sample_weight=None, zero_division='warn'))
# Recall 56.9%
print(recall_score(y_test, y_pred, labels=None, pos_label=1, average='micro',
                   sample_weight=None, zero_division='warn'))
# F1 Score 56.9%
print(f1_score(y_test, y_pred, labels=None, pos_label=1, average='micro',
               sample_weight=None, zero_division='warn'))
# F1 All classes
print(f1_score(y_test, y_pred, labels=None, pos_label=1, average=None,
               sample_weight=None, zero_division='warn'))

# Lasso (with PCA) 56.9%
clf = svm.LinearSVC(penalty='l1', loss='squared_hinge', dual=False, tol=0.0001,
    C=1, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None,
    verbose=0, random_state=None, max_iter=200000)
clf.fit(X_train_pca, y_train)
y_pred = clf.predict(X_test_pca)
print(clf.score(X_test_pca, y_test))
confusion_matrix(y_test, y_pred, labels=None, sample_weight=None, normalize=None)
plot_confusion_matrix(clf, X_test_pca, y_test)
plt.show()
print(clf.score(X_test_pca,y_test))

# Debate on Stack Exchange re train or not train for clustering

# K-Means Clustering 25%
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
model = KMeans(n_clusters=8)
y_pred = model.fit_predict(X_train)
print(y_pred)
centres = model.cluster_centers_
print('kmeans: {}'.format(silhouette_score(X_train, model.labels_,
                                           metric='euclidean')))
```

```python
import seaborn as sns
scores = [KMeans(n_clusters=i+2).fit(X_train).inertia_
          for i in range(10)]
sns.lineplot(x=np.arange(2, 12), y=scores)
plt.xlabel('Number of Clusters')
plt.ylabel("Inertia")
plt.title("Inertia of K-Means against Number of Clusters")

# https://towardsdatascience.com/unsupervised-learning-k-means-vs-hierarchical-clustering-5fe2da7c9
from scipy.cluster.hierarchy import dendrogram
from sklearn.cluster import AgglomerativeClustering

# https://scikit-learn.org/stable/auto_examples/cluster/plot_agglomerative_dendrogram.html#sphx-glr
def plot_dendrogram(model):
# Create linkage matrix and then plot the dendrogram
    # Create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack([model.children_, model.distances_,
                                      counts]).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix)
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None)
model = model.fit(X_train)
plt.title('Hierarchical Clustering Dendrogram for Activities')
# plot the top three levels of the dendrogram
plot_dendrogram(model)
plt.show()

# Random Forest
# https://scikit-learn.org/stable/modules/ensemble.html#forests-of-randomized-trees
# Single decision trees tend to overfit

# Are models over-fitting or under-fitting?
# Regularisation? Better training data split
# Choosing the best hyperparameters:
    # https://scikit-learn.org/stable/modules/grid_search.html#grid-search

# Asthma prediction using symptom tracking
# https://medium.com/vitalflo-health/the-power-of-remote-monitoring-through-machine-learning-ff067b
```