IBM ADVANCED DATA SCIENCE CAPSTONE PROJECT

ALEX BRAGA

IMU BASED SURFACE DETECTION

BASE DATASET

- Objective: Recognition of floor surface based on IMU data from robot
- Based on Kaggle CarrerCon 2019: Help Navigate robots competition
- https://www.kaggle.com/c/career-con-2019/data
- Data was collected from a robot in 9 different surfaces taking IMU data(10 channels), each data series have 128 measurements and refer to a single surface category
- Special thanks to Tampere University in Finland, Department of Signal Processing and Department of Automation and Mechanics Engineering for making the data available

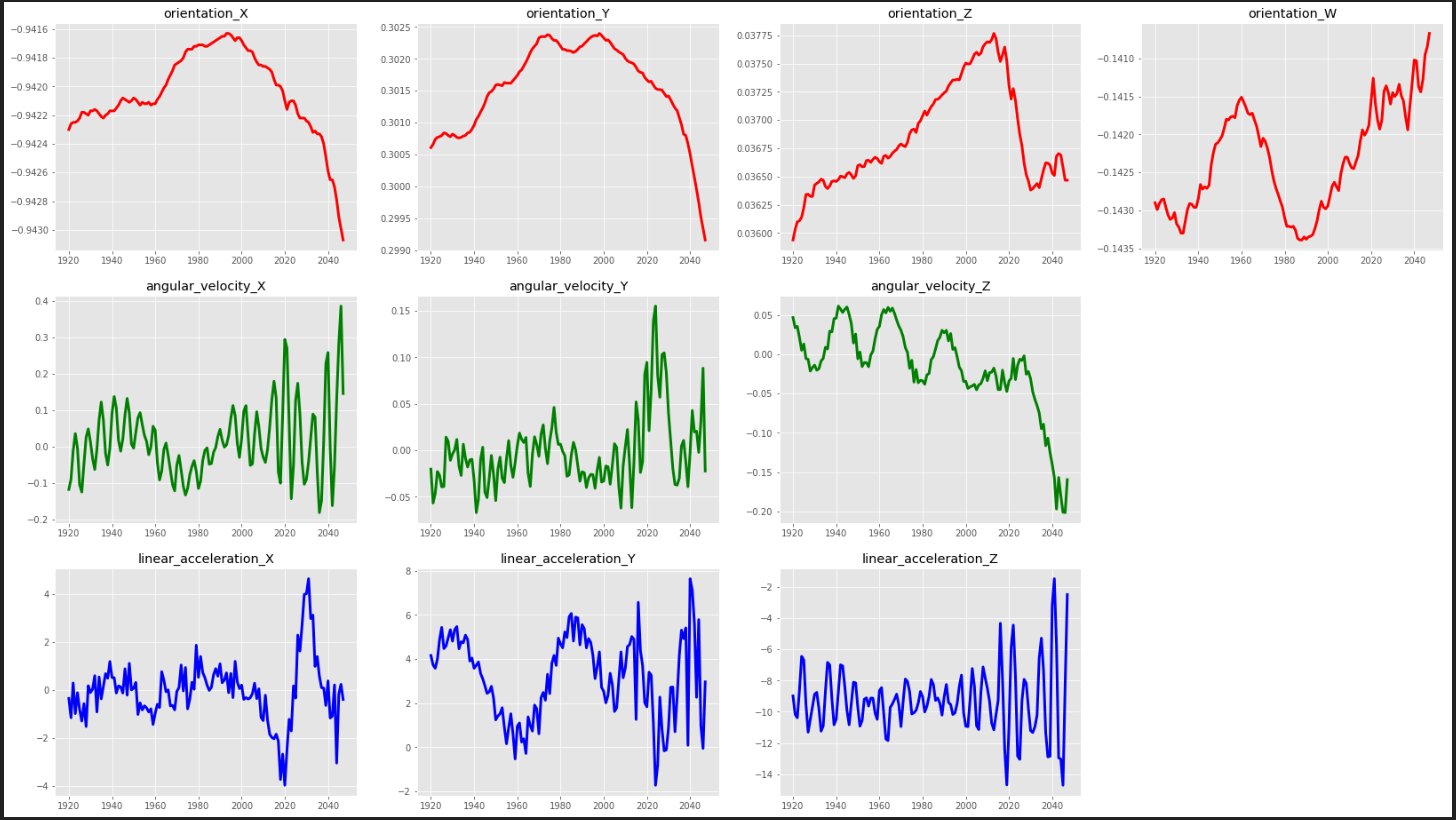
ROBOT SURFACE CLASSIFICATION

- Data consists of 2 files
- X
  - Row_ID: ID of a given row
  - Series_ID: ID of an 128 measurement series
  - Measurement_Number: Step of a measurement on a single series
  - Orientation X,Y,Z,W: Quaternion based location
  - Angular Velocity X,Y,Z: IMU date for rotation speed
  - Linear Acceleration X,Y,Z: IMU data for robot acceleration
- Y
  - Series ID: External reference to X data series
  - Group ID: Target ID for surface category
  - Surface: Surface Name

X DATA

```
X_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 487680 entries, 0 to 487679
Data columns (total 13 columns):
row_id                 487680 non-null object
series_id              487680 non-null int64
measurement_number     487680 non-null int64
orientation_X          487680 non-null float64
orientation_Y          487680 non-null float64
orientation_Z          487680 non-null float64
orientation_W          487680 non-null float64
angular_velocity_X     487680 non-null float64
angular_velocity_Y     487680 non-null float64
angular_velocity_Z     487680 non-null float64
linear_acceleration_X  487680 non-null float64
linear_acceleration_Y  487680 non-null float64
linear_acceleration_Z  487680 non-null float64
dtypes: float64(10), int64(2), object(1)
memory usage: 48.4+ MB
```



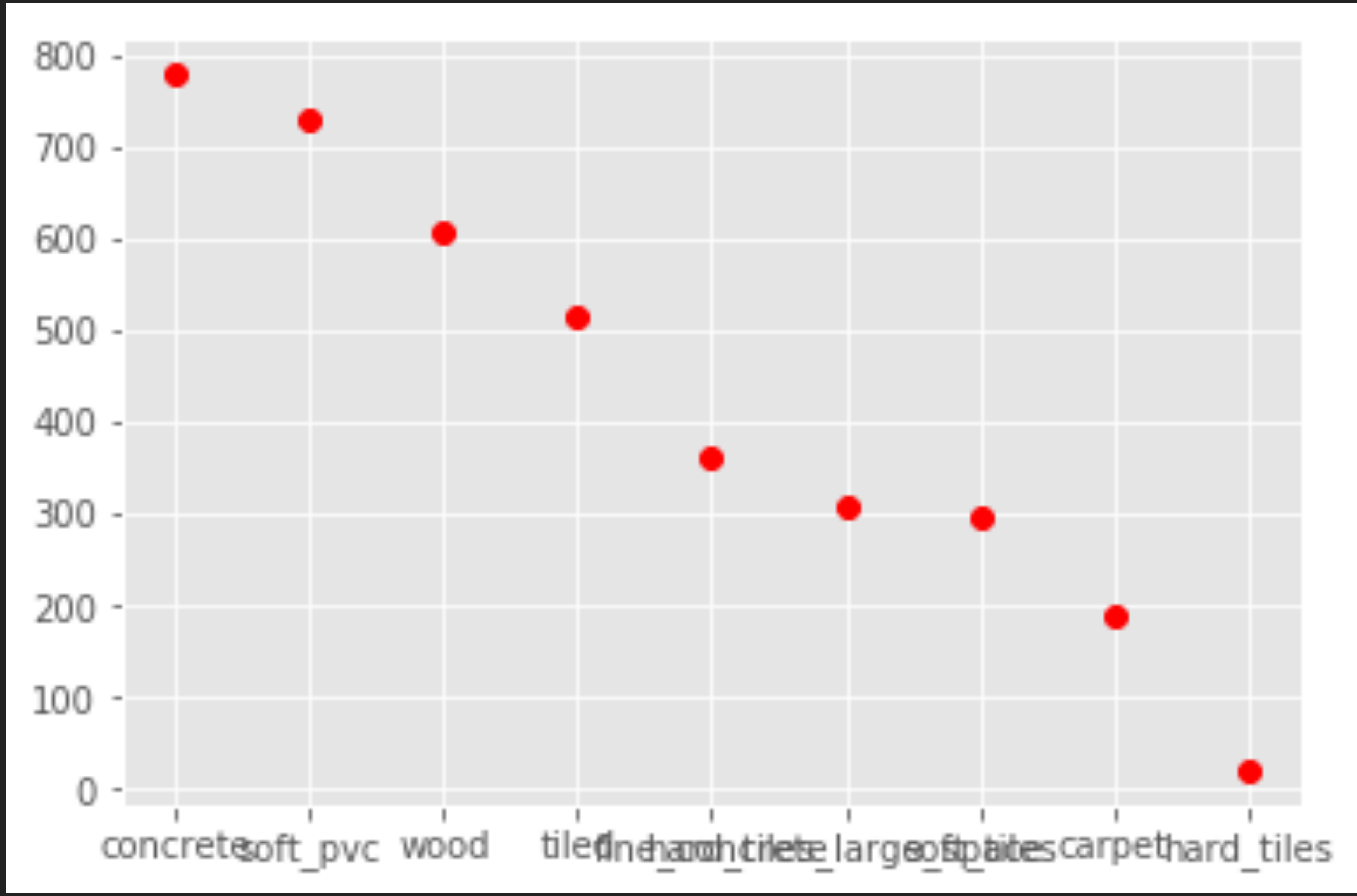
X DATA

```
Y_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3810 entries, 0 to 3809
Data columns (total 3 columns):
series_id     3810 non-null int64
group_id      3810 non-null int64
surface       3810 non-null object
dtypes: int64(2), object(1)
memory usage: 89.4+ KB
```

```
Y_data.head()
```

	series_id	group_id	surface
0	0	13	fine_concrete
1	1	31	concrete
2	2	20	concrete
3	3	31	concrete
4	4	22	soft_tiles



DATA CLEANING

```
X_data.isnull().sum()
```

```
row_id                  0
series_id               0
measurement_number      0
orientation_X           0
orientation_Y           0
orientation_Z           0
orientation_W           0
angular_velocity_X      0
angular_velocity_Y      0
angular_velocity_Z      0
linear_acceleration_X   0
linear_acceleration_Y   0
linear_acceleration_Z   0
dtype: int64
```

```
Y_data.isnull().sum()
```

```
series_id    0
group_id     0
surface      0
dtype: int64
```

```
X_data.duplicated().value_counts()
```

```
False    487680
dtype: int64
```

```
X_data.shape
```

```
(487680, 13)
```

```
Y_data.duplicated().value_counts()
```

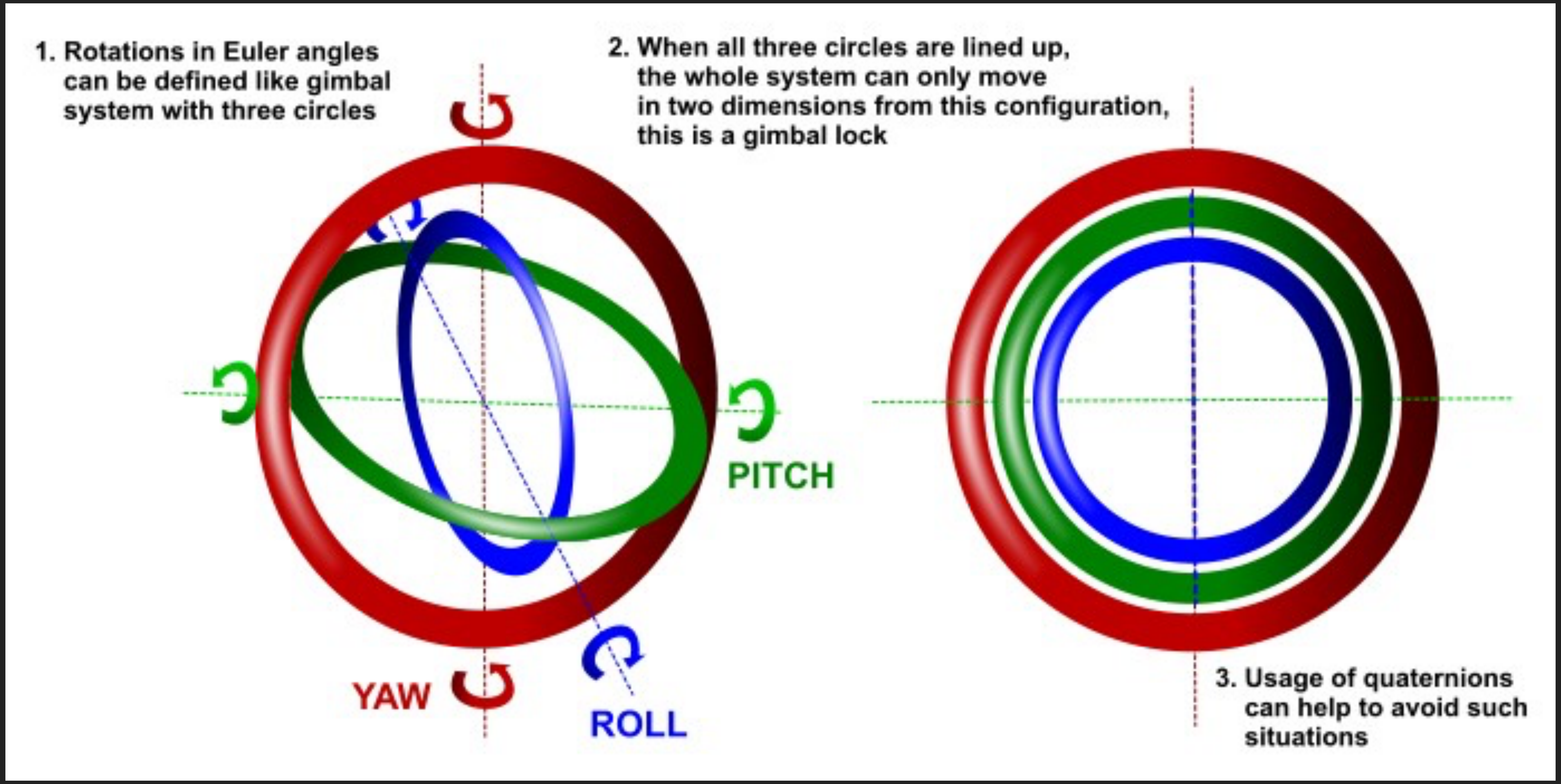
```
False    3810
dtype: int64
```

```
Y_data.shape
```

```
(3810, 3)
```

```
X_data.fillna(0, inplace = True)
X_data.replace(-np.inf, 0, inplace = True)
X_data.replace(np.inf, 0, inplace = True)
Y_data.fillna(0, inplace = True)
Y_data.replace(-np.inf, 0, inplace = True)
Y_data.replace(np.inf, 0, inplace = True)
```

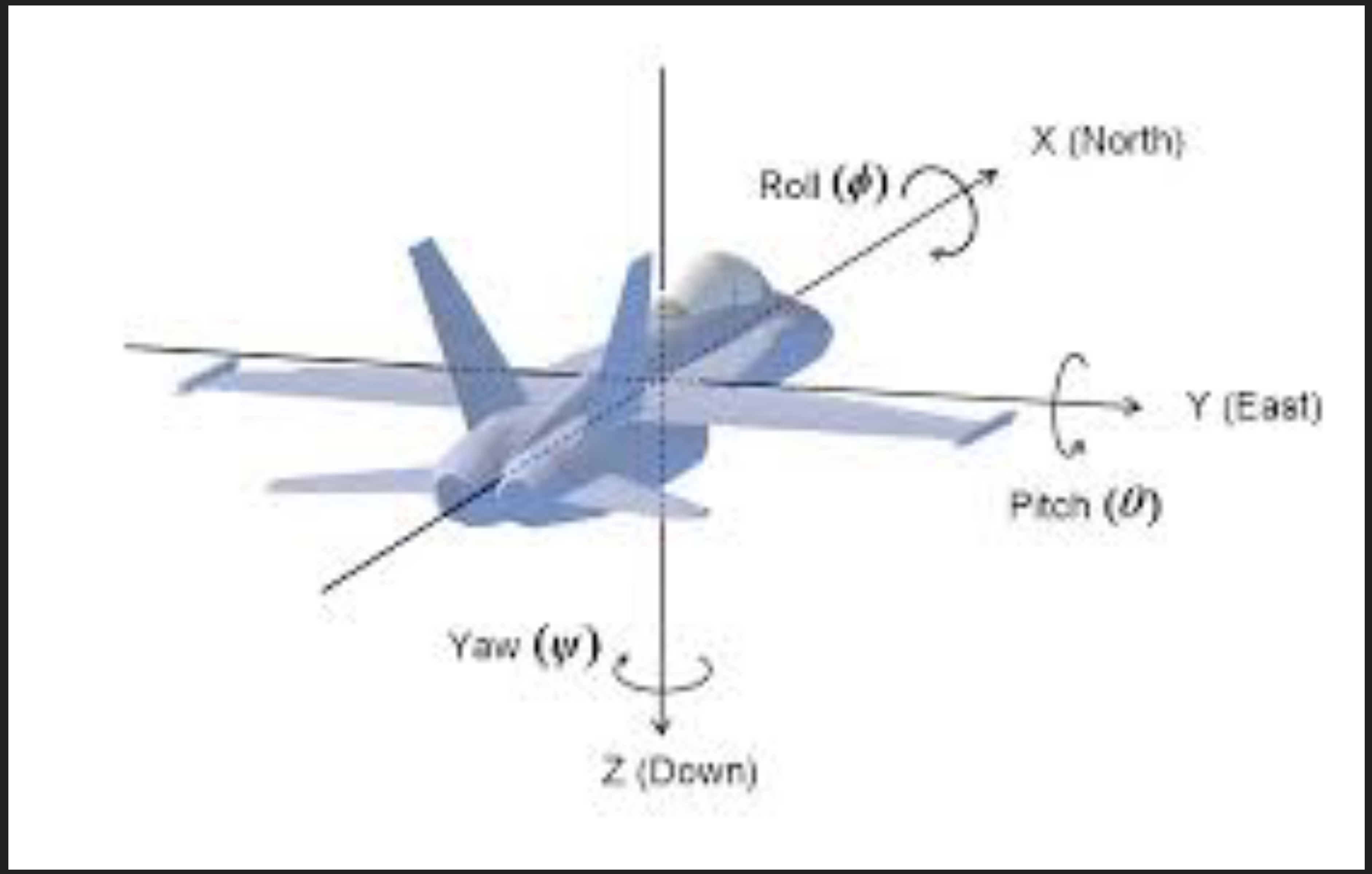
QUATERNION TO EULER



$$\phi = \arctan\left(\frac{2(ab+cd)}{a^2-b^2-c^2+d^2}\right),$$

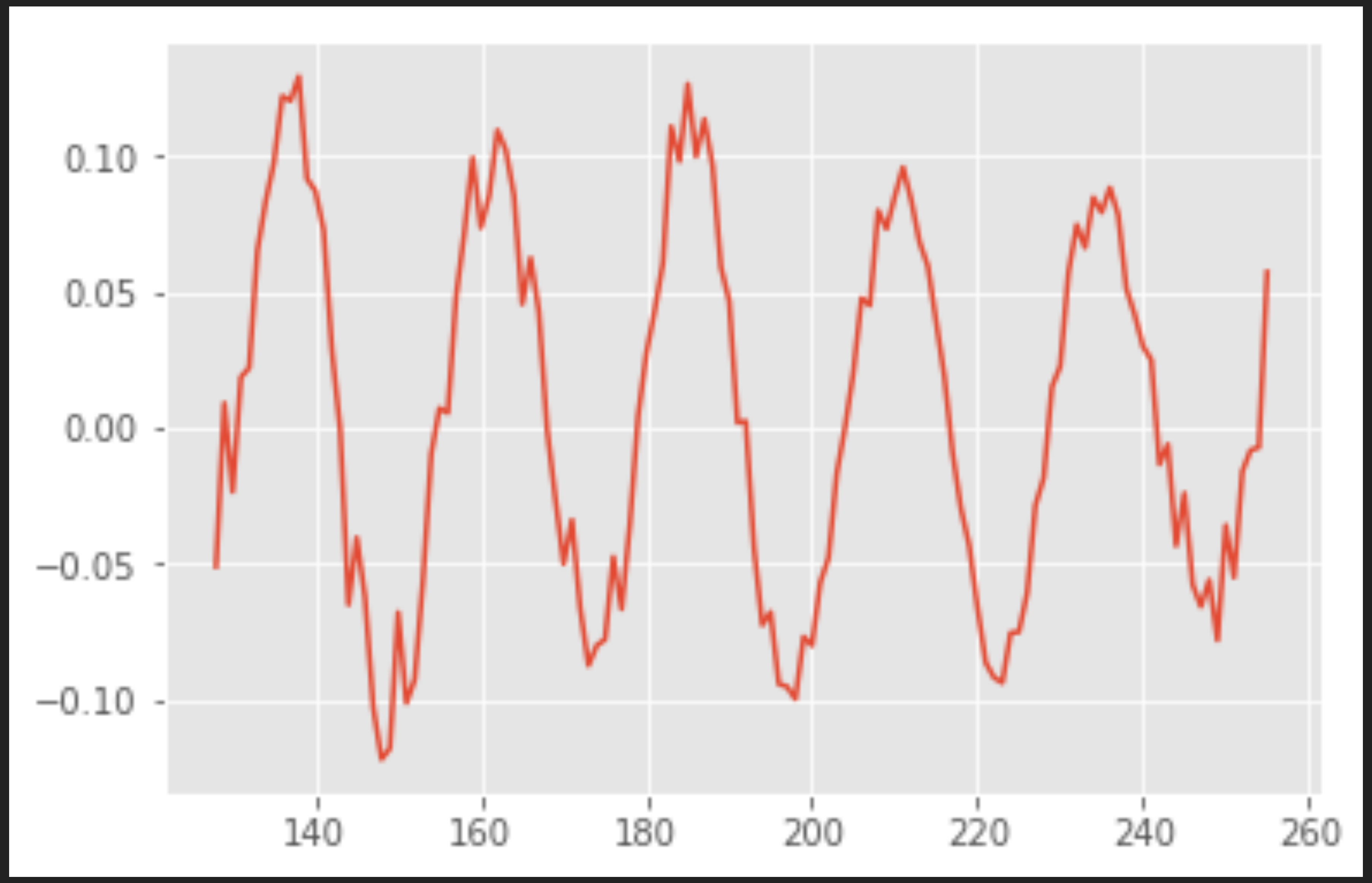
$$\theta = -\arcsin(2(bd-ac)), \text{ and}$$

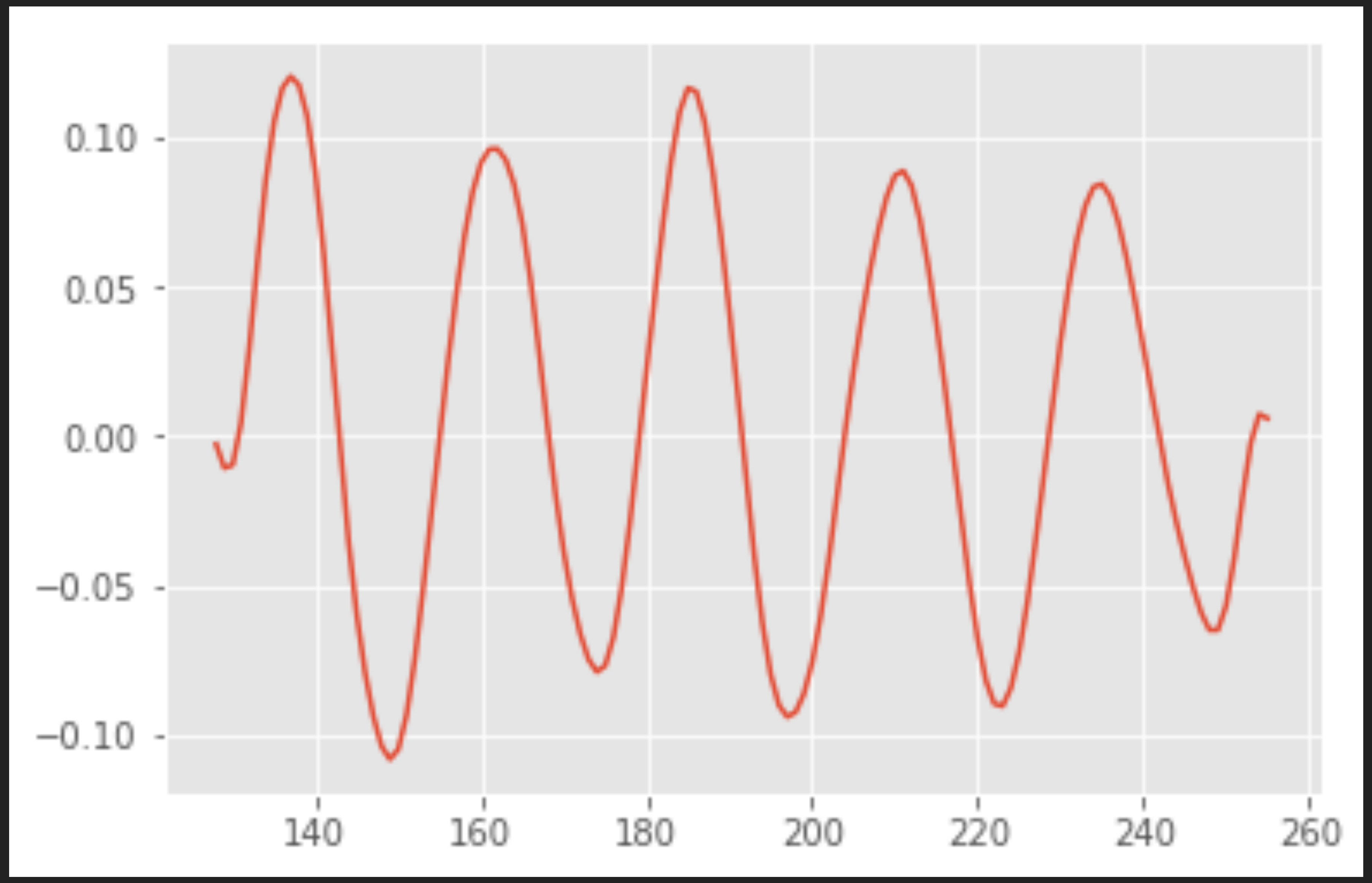
$$\psi = \arctan\left(\frac{2(ad+bc)}{a^2+b^2-c^2-d^2}\right).$$



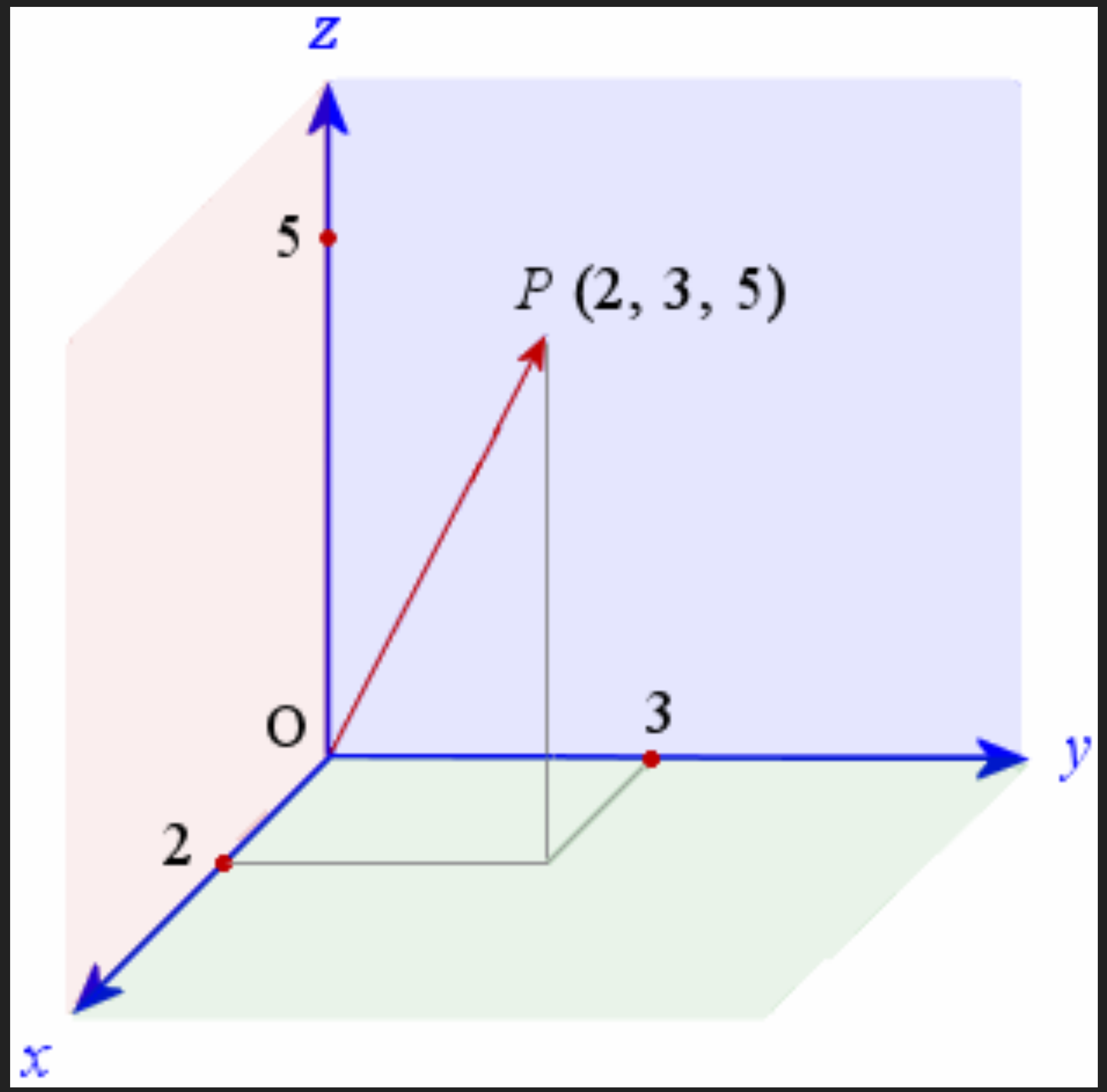
http://www.chrobotics.com/library/understanding-quaternions

NOISE REDUCTION





MAGNITUDE AND RELATION




```
X_data['total_angular_vel'] = (X_data['angular_velocity_X']**2 + X_data['angular_velocity_Y']**2 + X_data['angular_velocity_Z']**2)** 0.5
X_data['total_linear_acc'] = (X_data['linear_acceleration_X']**2 + X_data['linear_acceleration_Y']**2 + X_data['linear_acceleration_Z']**2)**0.5
X_data['total_orientation'] = (X_data['orientation_X']**2 + X_data['orientation_Y']**2 + X_data['orientation_Z']**2)**0.5
X_data['acc_vs_vel'] = X_data['total_linear_acc'] / X_data['total_angular_vel']
X_data['total_angle'] = (X_data['euler_x'] ** 2 + X_data['euler_y'] ** 2 + X_data['euler_z'] ** 2) ** 0.5
X_data['angle_vs_acc'] = X_data['total_angle'] / X_data['total_linear_acc']
X_data['angle_vs_vel'] = X_data['total_angle'] / X_data['total_angular_vel']
```

FROM TIME TO MEASUREMENT DOMAIN

Series	Measurement	X	Y	Z	…
1	1				
1	2				
1	3				
1	…				
1	128				
2	1				
2	2				
2	3				
2	…				
2	128				
…	…				

DATA

Series	X_mean	X_max	X_min	…
1				
2				
3				
…				

DATA

```
x = Input.values
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
Input = pd.DataFrame(x_scaled)
```

LABEL ENCODING

```
Y_data.head()
```

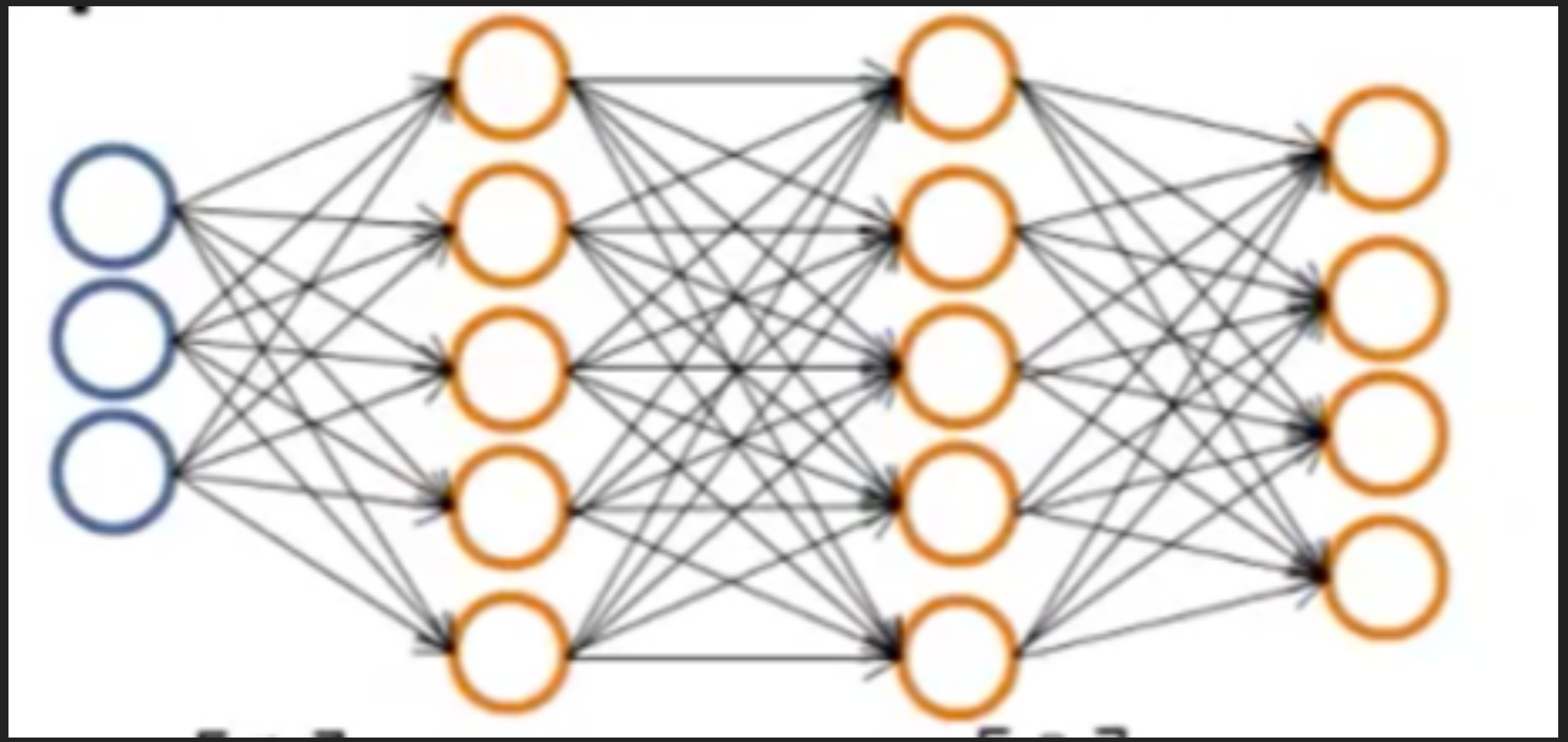
	series_id	group_id	surface
0	0	13	fine_concrete
1	1	31	concrete
2	2	20	concrete
3	3	31	concrete
4	4	22	soft_tiles

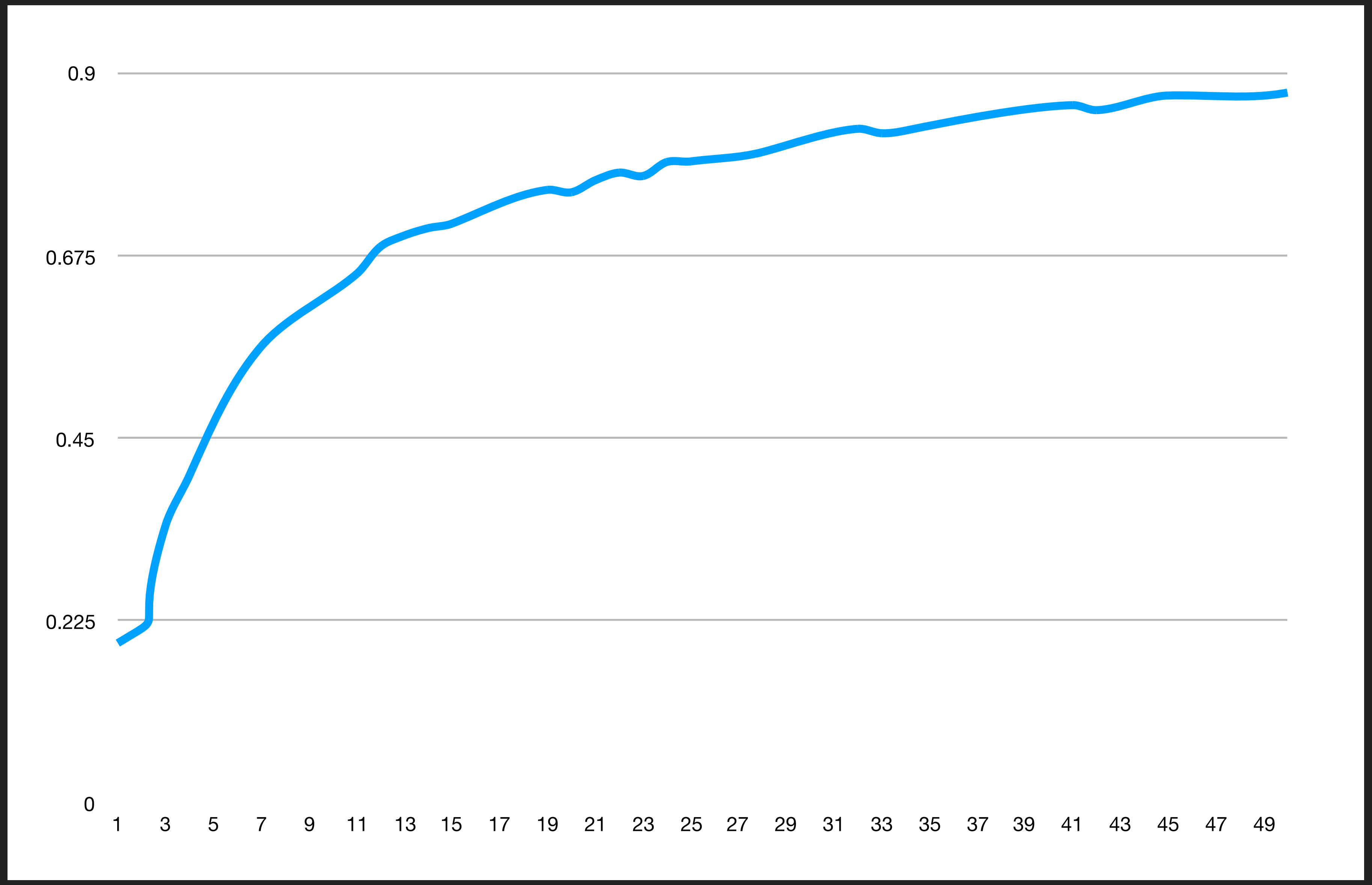
	series_id	group_id	surface
0	0	13	2
1	1	31	1
2	2	20	1
3	3	31	1
4	4	22	6

	0	1	2	3	4	5	6	7	8
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

MODEL

SEQUENTIAL NEURAL NETWORK





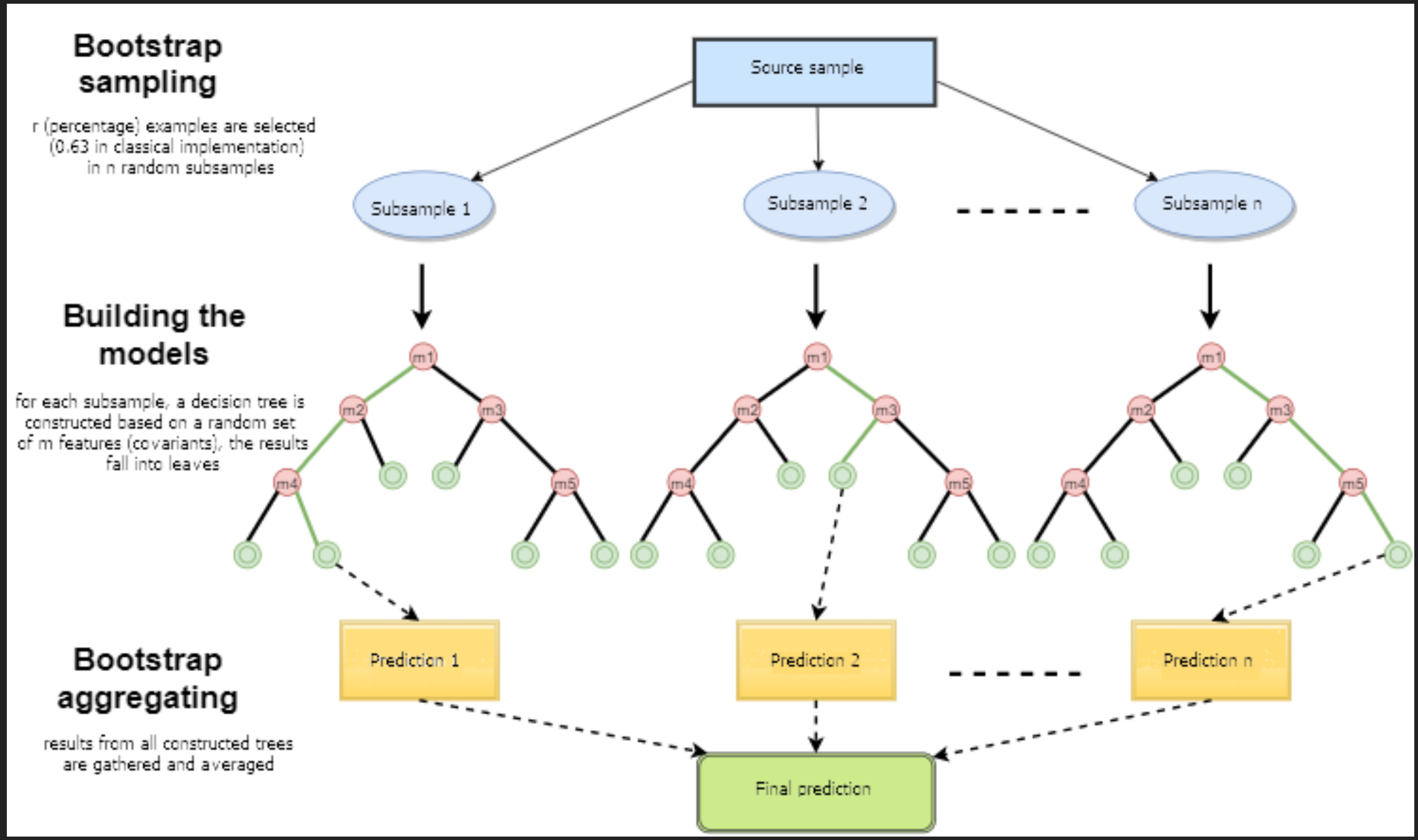
Let's divide the data in test and train using 80/20 ratio given then small dataset.

```
xTrain, xTest, yTrain, yTest = train_test_split(Input, Y_data, test_size=0
.2, random_state = 0)
```

```
model = keras.Sequential()
model.add(keras.layers.Dense(64, input_shape=(100,), activation=tf.nn.sigmoid))
model.add(keras.layers.Dense(32,activation=tf.nn.sigmoid))
model.add(keras.layers.Dense(16,activation=tf.nn.sigmoid))
model.add(keras.layers.Dense(9,activation=tf.nn.softmax))

model.compile(optimizer=tf.train.RMSPropOptimizer(0.01),
              loss=tf.keras.losses.categorical_crossentropy,
              metrics=[tf.keras.metrics.categorical_accuracy])

model.fit(xTrain, yTrainOHE, epochs=50)
```

```
model.evaluate(xTest,yTestOHE)

762/762 [==============================] - 0s 85us/sample - loss: 0.5579
- categorical_accuracy: 0.8163
```

RANDOM FOREST




```
folds = StratifiedKFold(n_splits=5, shuffle=True, random_state=60)
predicted = np.zeros((xTest.shape[0],9))
measured= np.zeros((Input.shape[0]))
score = 0

for times, (trn_idx, val_idx) in enumerate(folds.split(Input.values,Y_data
['surface'].values)):
    model2 = RandomForestClassifier(n_estimators=300, max_depth=5, min_sam
ples_split=5, n_jobs=-1)
    model2.fit(Input.iloc[trn_idx],Y_data['surface'][trn_idx])
    measured[val_idx] = model2.predict(Input.iloc[val_idx])
    predicted += model2.predict_proba(xTest)/folds.n_splits
    score += model2.score(Input.iloc[val_idx],Y_data['surface'][val_idx])
    print("Fold: {} score: {}".format(times,model2.score(Input.iloc[val_id
x],Y_data['surface'][val_idx])))

    gc.collect()
```

```
Fold: 0 score: 0.7023498694516971
Fold: 1 score: 0.7633986928104575
Fold: 2 score: 0.6968503937007874
Fold: 3 score: 0.6934210526315789
Fold: 4 score: 0.7357992073976222
```

```
print('Average score', score / folds.n_splits)
```

```
Average score 0.7183638431984287
```

ADDITIONAL FEATURES

```
Input2=Input.copy()
for col in X_data.columns:
        if col in ['row_id','series_id','measurement_number']:
            continue
        print ("FE on column ", col, "...")
        Input2[col + '_mad'] = X_data.groupby(['series_id'])[col].apply(lambda x: np.median(np.abs(np.diff(x))))
        Input2[col + '_abs_max'] = X_data.groupby(['series_id'])[col].apply(lambda x: np.max(np.abs(x)))
        Input2[col + '_abs_min'] = X_data.groupby(['series_id'])[col].apply(lambda x: np.min(np.abs(x)))
        Input2[col + '_abs_avg'] = (Input2[col + '_abs_min'] + Input2[col + '_abs_max'])/2
        Input2[col + '_skew'] = X_data.groupby(['series_id'])[col].skew()
        Input2[col + '_mad'] = X_data.groupby(['series_id'])[col].mad()
        Input2[col + '_q25'] = X_data.groupby(['series_id'])[col].quantile(0.25)
        Input2[col + '_q75'] = X_data.groupby(['series_id'])[col].quantile(0.75)
        Input2[col + '_q95'] = X_data.groupby(['series_id'])[col].quantile(0.95)
        Input2[col + '_iqr'] = Input2[col + '_q75'] - Input2[col + '_q25']
        Input2[col + '_SSC'] = X_data.groupby(['series_id'])[col].apply(SSC)
        Input2[col + '_skewness'] = X_data.groupby(['series_id'])[col].apply(skewness)
        Input2[col + '_wave_lenght'] = X_data.groupby(['series_id'])[col].apply(wave_length)
        Input2[col + '_norm_entropy'] = X_data.groupby(['series_id'])[col].apply(norm_entropy)
        Input2[col + '_SRAV'] = X_data.groupby(['series_id'])[col].apply(SRAV)
        Input2[col + '_kurtosis'] = X_data.groupby(['series_id'])[col].apply(_kurtosis)
        Input2[col + '_zero_crossing'] = X_data.groupby(['series_id'])[col].apply(zero_crossing)
```

NEURAL NETWORK

```
model3 = keras.Sequential()
model3.add(keras.layers.Dense(256, input_shape=(420,), activation=tf.nn.si
gmoid))
model3.add(keras.layers.Dense(128,activation=tf.nn.sigmoid))
model3.add(keras.layers.Dense(64,activation=tf.nn.sigmoid))
model3.add(keras.layers.Dense(32,activation=tf.nn.sigmoid))
model3.add(keras.layers.Dense(9,activation=tf.nn.softmax))

model3.compile(loss='categorical_crossentropy', optimizer='adam', metrics=
['accuracy'])


model3.fit(xTrain2, yTrainOHE2, epochs=200)
```

```
model3.evaluate(xTest2,yTestOHE2)
```

```
762/762 [==============================] - 0s 126us/sample - loss: 0.3634
- acc: 0.9094

[0.3634368342364554, 0.9094488]
```

RANDOM FOREST

```
folds2 = StratifiedKFold(n_splits=10, shuffle=True, random_state=60)
predicted2 = np.zeros((xTest.shape[0],9))
measured2= np.zeros((Input.shape[0]))
score2 = 0

for times, (trn_idx, val_idx) in enumerate(folds2.split(Input2.values,Y_da
ta2['surface'].values)):
    model4 = RandomForestClassifier(n_estimators=700, max_depth=20, min_sa
mples_split=5, n_jobs=-1)
    model4.fit(Input2.iloc[trn_idx],Y_data2['surface'][trn_idx])
    measured2[val_idx] = model4.predict(Input2.iloc[val_idx])
    predicted2 += model4.predict_proba(xTest2)/folds2.n_splits
    score2 += model4.score(Input2.iloc[val_idx],Y_data2['surface'][val_idx
])
    print("Fold: {} score: {}".format(times,model4.score(Input2.iloc[val_i
dx],Y_data2['surface'][val_idx])))

    gc.collect()
```

```
print('Average score', score2 / folds2.n_splits)
```

```
Average score 0.9097169076841908
```

THANKS

Alex Braga