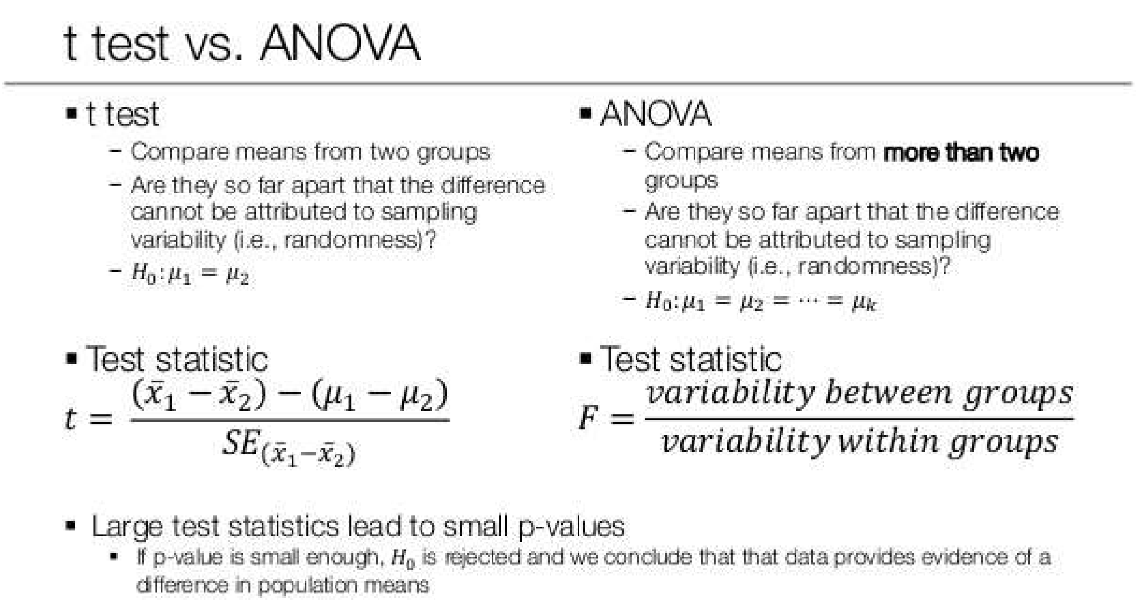
Feature Selection Based on Univariate (ANOVA) Test for Classification

What is Univariate (ANOVA) Test

The elimination process aims to reduce the size of the input feature set and at the same time to retain the class discriminatory information for classification problems.



An F-test is any statistical test in which the test statistic has an F-distribution under the null hypothesis.

Analysis of variance (ANOVA) is a collection of statistical models and their associated estimation procedures (such as the "variation" among and between groups) used to analyze the differences among group means in a sample.

The F-test is used for comparing the factors of the total deviation. For example, in one-way, or single-factor ANOVA, statistical significance is tested for by comparing the F test statistic

$$F = \frac{\text{variance between treatments}}{\text{variance within treatments}}$$

$$F = \frac{MS_{\text{Treatments}}}{MS_{\text{Error}}} = \frac{SS_{\text{Treatments}}/(I-1)}{SS_{\text{Error}}/(n_T - I)}$$

`sklearn.feature_selection`: Feature Selection

The `sklearn.feature_selection` module implements feature selection algorithms. It currently includes univariate filter selection methods and the recursive feature elimination algorithm.

User guide: See the Feature selection section for further details.

`feature_selection.GenericUnivariateSelect` ([…])	Univariate feature selector with configurable strategy.
`feature_selection.SelectPercentile` ([…])	Select features according to a percentile of the highest scores.
`feature_selection.SelectKBest` ([score_func, k])	Select features according to the k highest scores.
`feature_selection.SelectFpr` ([score_func, alpha])	Filter: Select the pvalues below alpha based on a FPR test.
`feature_selection.SelectFdr` ([score_func, alpha])	Filter: Select the p-values for an estimated false discovery rate
`feature_selection.SelectFromModel` (estimator)	Meta-transformer for selecting features based on importance weights.
`feature_selection.SelectFwe` ([score_func, alpha])	Filter: Select the p-values corresponding to Family-wise error rate
`feature_selection.RFE` (estimator[, …])	Feature ranking with recursive feature elimination.
`feature_selection.RFECV` (estimator[, step, …])	Feature ranking with recursive feature elimination and cross-validated selection of the best number of features.
`feature_selection.VarianceThreshold` ([threshold])	Feature selector that removes all low-variance features.
`feature_selection.chi2` (X, y)	Compute chi-squared stats between each non-negative feature and class.
`feature_selection.f_classif` (X, y)	Compute the ANOVA F-value for the provided sample.
`feature_selection.f_regression` (X, y[, center])	Univariate linear regression tests.
`feature_selection.mutual_info_classif` (X, y)	Estimate mutual information for a discrete target variable.
`feature_selection.mutual_info_regression` (X, y)	Estimate mutual information for a continuous target variable.

Classification Problem

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.feature_selection import VarianceThreshold
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import f_classif, f_regression
from sklearn.feature_selection import SelectKBest, SelectPercentile
```

In [3]:
```
data = pd.read_csv('0.9_5subjectslabelled_data.csv', nrows = 31437)
data.head()
```

Out[3]:

	Time Snap	AccX	AccY	AccZ	Gyro_X	Knee Angles	Gait Cycle Phase
0	120.775	-0.181472	-0.088708	-0.665352	0.087145	67.223821	5
1	120.780	-0.181443	-0.088745	-0.659870	0.103506	67.217858	5
2	120.785	-0.183826	-0.089735	-0.654215	0.117635	67.154903	5
3	120.790	-0.188545	-0.091706	-0.648504	0.129259	67.011479	5
4	120.795	-0.195535	-0.094645	-0.642857	0.138193	66.799616	5

In [4]:
```
X = data.drop('Gait Cycle Phase', axis = 1)
y = data['Gait Cycle Phase']

X.shape, y.shape
```

Out[4]:
```
((31436, 6), (31436,))
```

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ran
```

Remove Constant, Quasi Constant, and Correlated Features

In [6]:
```
#remove constant and quasi constant features
constant_filter = VarianceThreshold(threshold=0.01)
constant_filter.fit(X_train)
X_train_filter = constant_filter.transform(X_train)
X_test_filter = constant_filter.transform(X_test)
```

In [7]:
```
X_train_filter.shape, X_test_filter.shape
```

Out[7]:
```
((25148, 6), (6288, 6))
```

In [8]:
```
#remove duplicate features
X_train_T = X_train_filter.T
X_test_T = X_test_filter.T
```

In [9]:
```
X_train_T = pd.DataFrame(X_train_T)
X_test_T = pd.DataFrame(X_test_T)
```

In [10]:
```
X_train_T.duplicated().sum()
```

Out[10]:
```
0
```

In [11]:
```
duplicated_features = X_train_T.duplicated()
```

In [12]:
```
features_to_keep = [not index for index in duplicated_features]

X_train_unique = X_train_T[features_to_keep].T
X_test_unique = X_test_T[features_to_keep].T
```

In [13]:
```
X_train_unique.shape, X_train.shape
```

Out[13]:
```
((25148, 6), (25148, 6))
```

Now do F-Test

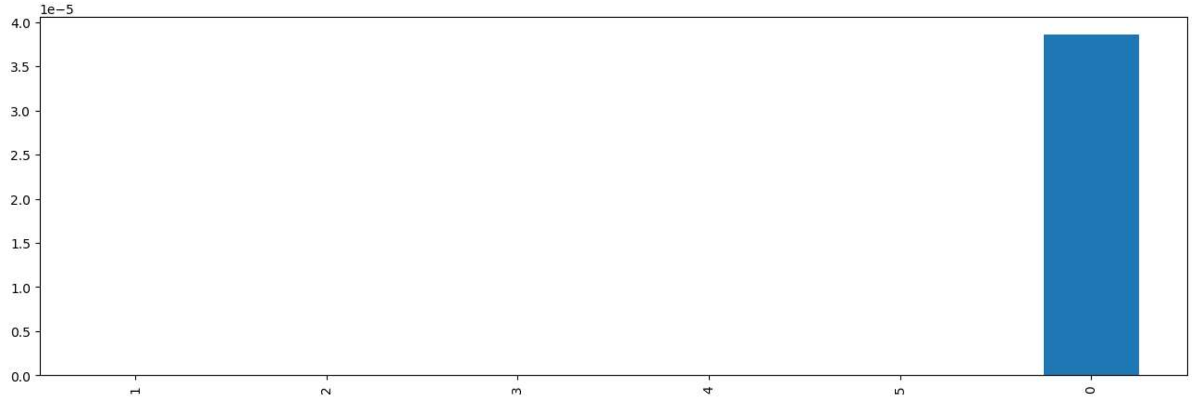
In [14]:
```
sel = f_classif(X_train_unique, y_train)
sel
```

Out[14]:
```
(array([5.00935732e+00, 1.90286745e+03, 4.25483808e+02, 8.93406758e+02,
        4.60885635e+02, 1.41982826e+04]),
 array([3.86162677e-05, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00]))
```

In [15]:
```
p_values = pd.Series(sel[1])
p_values.index = X_train_unique.columns
p_values.sort_values(ascending = True, inplace = True)
```

In [16]:
```
p_values.plot.bar(figsize = (16, 5))
```

Out[16]: <Axes: >



In [17]:
```
p_values
```

Out[17]:
```
1    0.000000
2    0.000000
3    0.000000
4    0.000000
5    0.000000
0    0.000039
dtype: float64
```

In [18]:
```
p_values = p_values[p_values<0.05]
```

In [26]:
```
selected_features = p_values[p_values < 0.05]
selected_feature_names = selected_features.index.tolist()
print(selected_feature_names)
```

```
[1, 2, 3, 4, 5, 0]
```

In [19]:
```
p_values.index
```

Out[19]: `Int64Index([1, 2, 3, 4, 5, 0], dtype='int64')`

In [24]:
```
index_labels = p_values.index
label_of_index_zero = index_labels[0]
print(label_of_index_zero)
```

```
1
```

In [20]:
```
X_train_p = X_train_unique[p_values.index]
X_test_p = X_test_unique[p_values.index]
```

Build the classifiers and compare the performance

In [21]:
```
def run_randomForest(X_train, X_test, y_train, y_test):
    clf = RandomForestClassifier(n_estimators=100, random_state=0, n_jobs = -1
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print('Accuracy: ', accuracy_score(y_test, y_pred))
```

In [22]:
```
%%time
run_randomForest(X_train_p, X_test_p, y_train, y_test)
```

```
Accuracy:  0.9217557251908397
CPU times: total: 34.1 s
Wall time: 9.06 s
```

In [23]:
```
%%time
run_randomForest(X_train, X_test, y_train, y_test)
```

```
Accuracy:  0.9212786259541985
CPU times: total: 34.5 s
Wall time: 8.5 s
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: