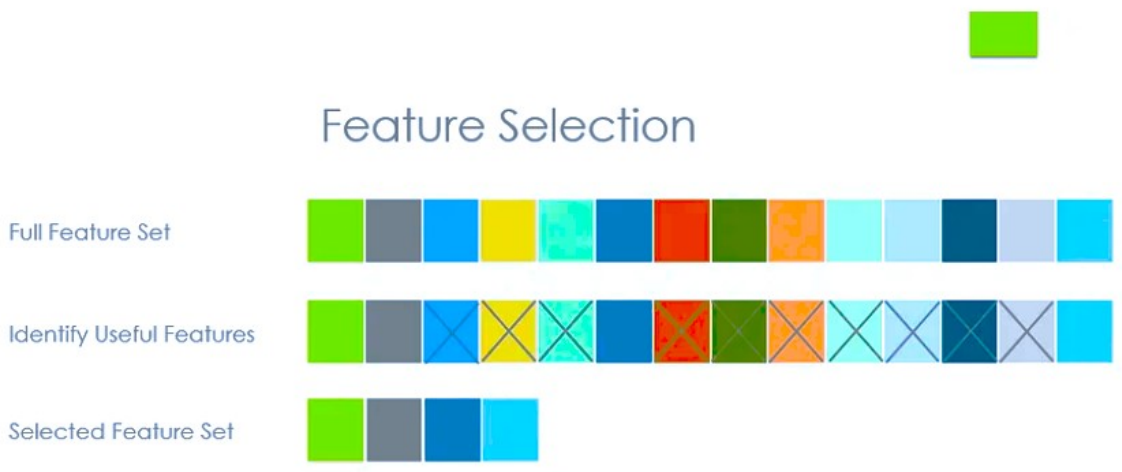
# What is Feature Selection?

Feature selection is a process in machine learning and data analysis where a subset of relevant features or **variables is selected from a larger set of available features**. The goal of feature selection is to **identify the most informative and relevant features that contribute the most** to the predictive performance of a model while reducing complexity, improving accuracy, and mitigating the risk of overfitting.



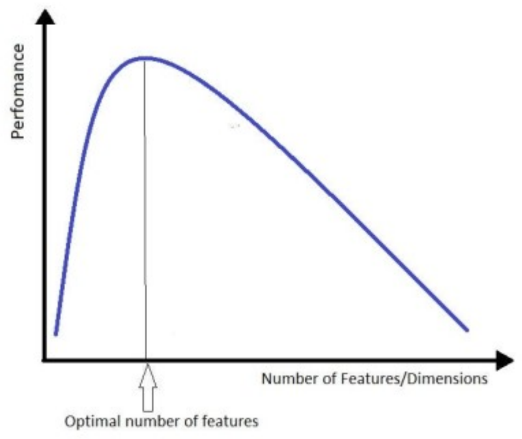
## The following are the primary reasons for performing feature selection:

- **Simplifying the model**: By reducing the number of features, the complexity of the model can be reduced, leading to simpler and more interpretable models.
- **Improving prediction performance**: Irrelevant or redundant features can introduce noise and lead to overfitting. Feature selection helps to focus on the most informative features, leading to improved generalization and prediction accuracy.
- **Reducing computational complexity**: By selecting a smaller subset of features, the computational resources required for model training and prediction can be reduced, resulting in faster processing times.
- **Enhancing interpretability**: Feature selection can help identify the most influential features, enabling better understanding and interpretation of the relationships between the features and the target variable.

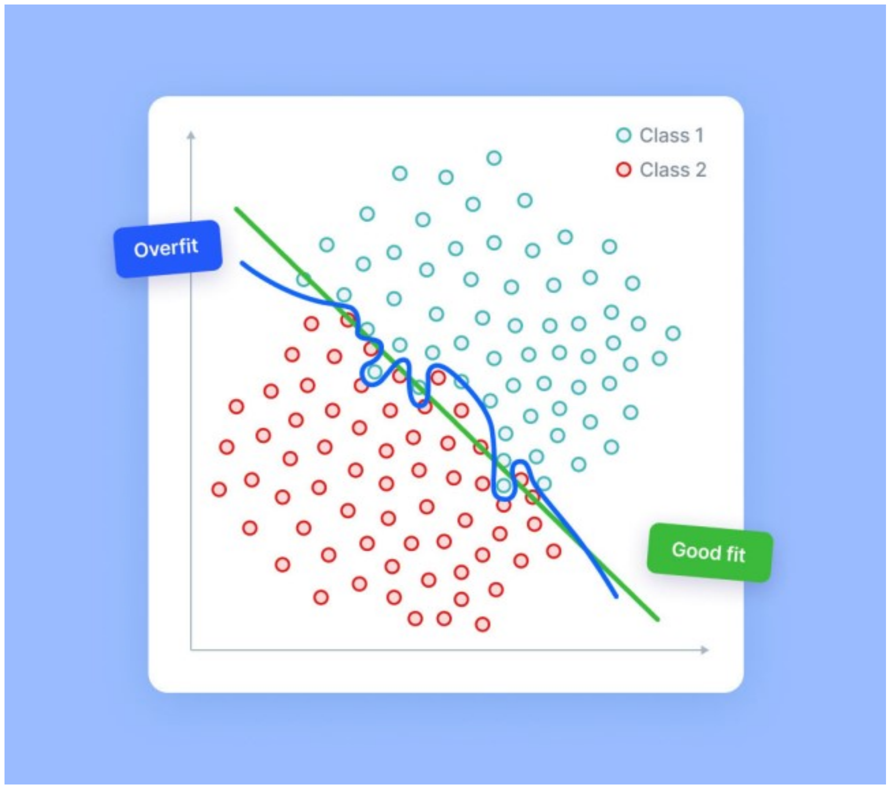
## What problems does Feature Selection solve?

**Feature selection can help solve various problems in machine learning and data analysis.**

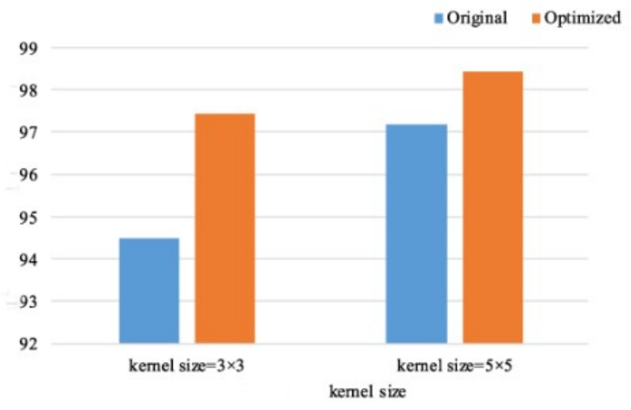
1. **Curse of Dimensionality**: When dealing with high-dimensional data, where the number of features is large compared to the number of observations, feature selection can alleviate the curse of dimensionality. By selecting a subset of relevant features, the data representation becomes more focused, improving the efficiency and performance of learning algorithms.



2. **Overfitting**: Including irrelevant or redundant features in a model can lead to overfitting, where the model becomes overly complex and performs poorly on unseen data. Feature selection helps to identify and remove such features, reducing the risk of overfitting and improving the generalization ability of the model.

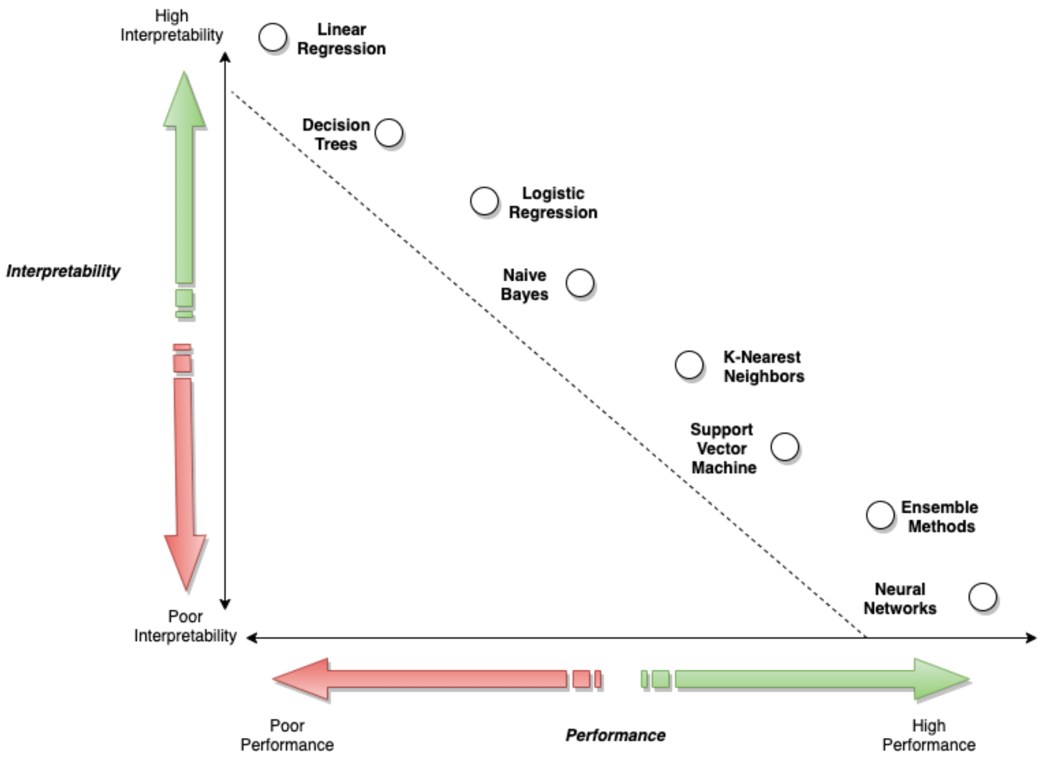


3. **Computational Efficiency**: In situations where computational resources are limited, such as working with large datasets, feature selection can significantly reduce the computational complexity. By selecting a smaller subset of features, the training and inference times of the model can be reduced, enabling faster processing.

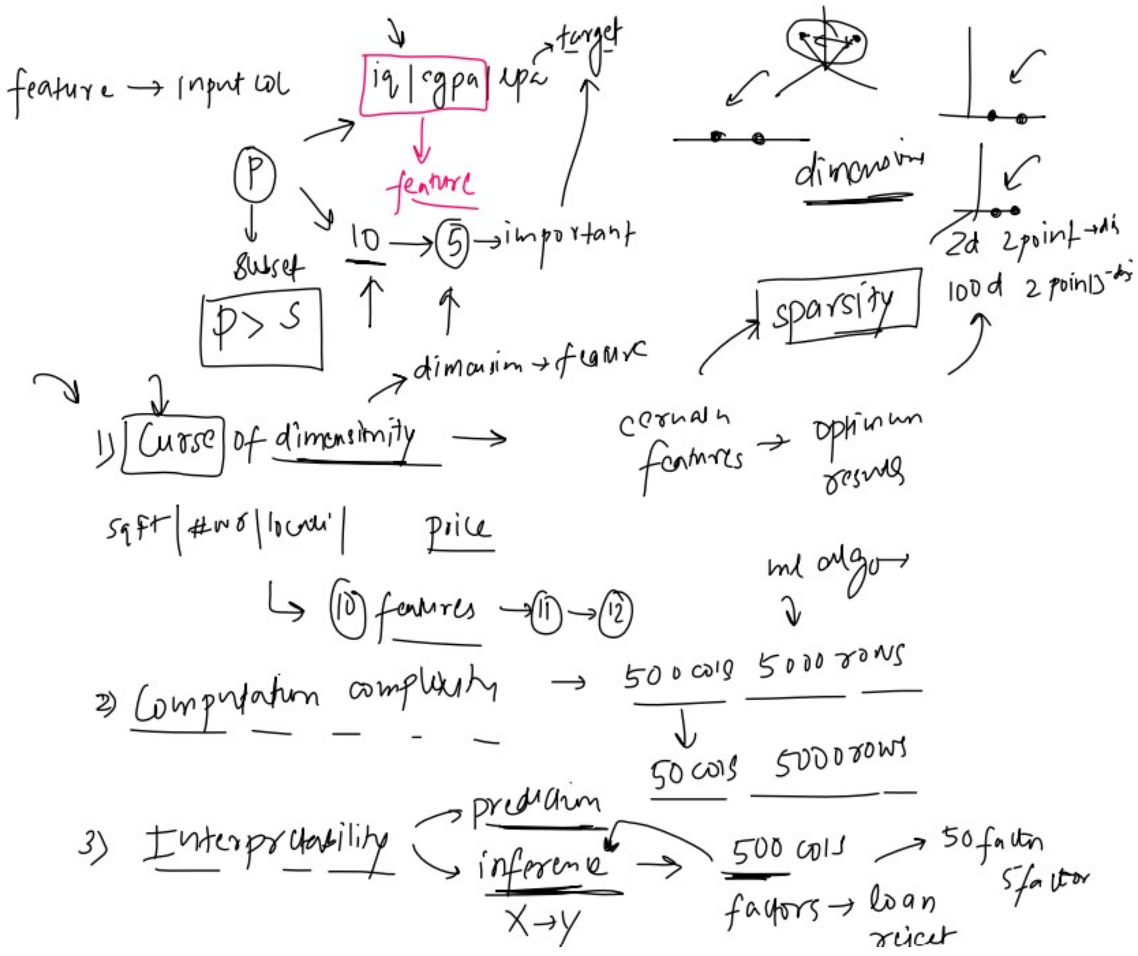


Computational efficiency after utilizing the optimization

4. **Interpretability**: Feature selection can improve the interpretability of models by identifying the most relevant features. This is particularly important in fields where model transparency and explainability are crucial, such as healthcare or finance. By selecting a subset of interpretable features, it becomes easier to understand the factors influencing the model's predictions.

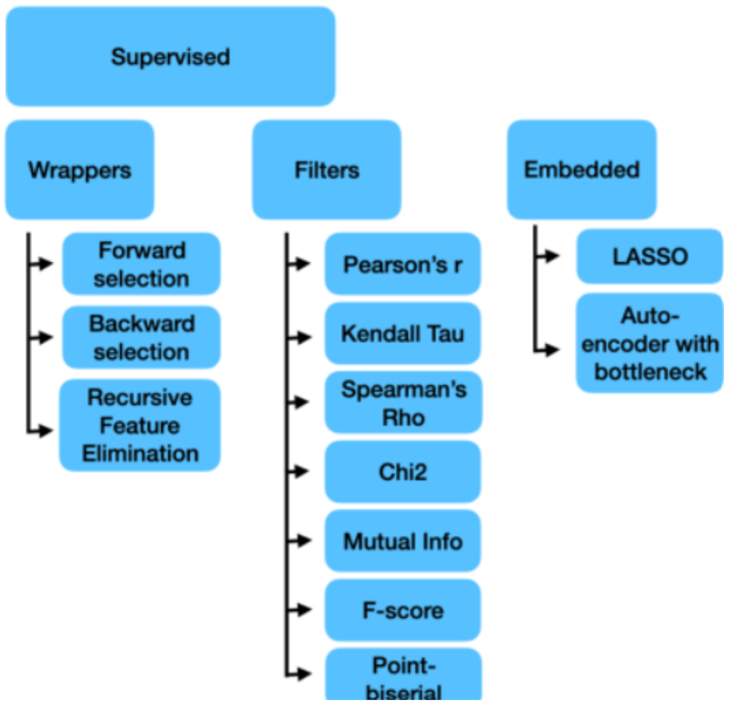


**Explanation :**



## Types Of Feature Selection Methods

- **1.Filter Method**
- **2.Wrapper Method**
- **3.Embedded Method**



## 1. Filter Based Feature Selection

Filter-based feature selection techniques are methods that use statistical measures to score each feature independently, and then select a subset of features based on these scores. These methods are called **"filter"** methods because they essentially **filter out the features that do not meet some criterion**.

**Here are a few commonly used filter-based feature selection methods:**

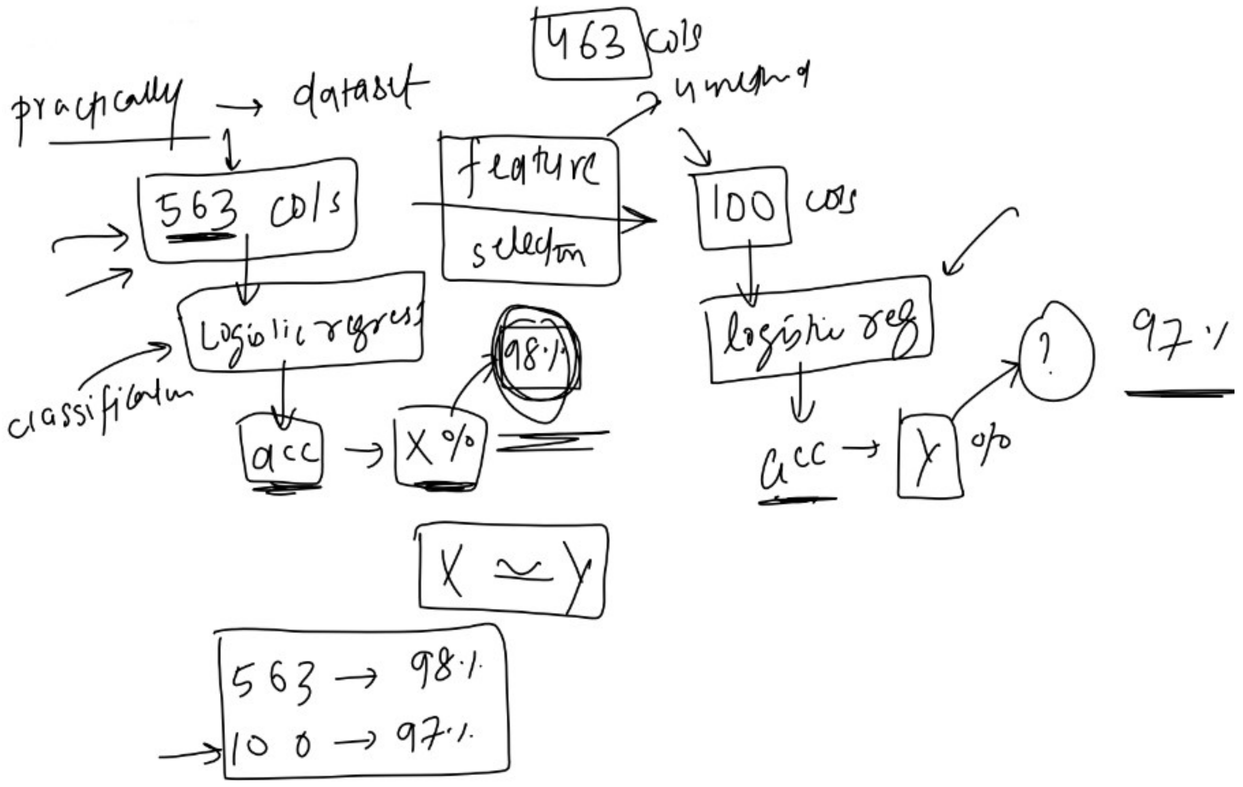
1. **Correlation-based Feature Selection**: This method measures the correlation between each feature and the target variable. Features with higher correlation values are considered more relevant and are selected for inclusion. Common correlation measures include Pearson correlation coefficient for continuous variables and point biserial correlation for categorical variables.

2. **Information Gain**: Information gain is a measure from information theory that quantifies the amount of information that a feature provides about the target variable. Features with higher information gain are considered more informative and are selected. This method is commonly used for feature selection in decision tree-based algorithms.

3. **Chi-Square Test**: The chi-square test is used for feature selection when dealing with categorical target variables. It measures the independence between each feature and the target variable by comparing the observed and expected frequencies in a contingency table. Features with significant chi-square test statistics are selected.

4. **ANOVA**: Analysis of Variance (ANOVA) is used when the target variable is continuous and the features are categorical. It compares the means of each feature across different levels of the target variable and determines if there are significant differences. Features with significant differences are selected.

**Pratical**



In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
df = pd.read_csv("D:\\datascience\\Nitish sir\\data Preprocessing\\train.csv").drop(columns='subject')
df.head()

# Data is Already Scaled , it has 562 Columns
```

Out[2]:

| | tBodyAcc-mean()-X | tBodyAcc-mean()-Y | tBodyAcc-mean()-Z | tBodyAcc-std()-X | tBodyAcc-std()-Y | tBodyAcc-std()-Z | tBodyAcc-mad()-X | tBodyAcc-mad()-Y | tBodyAcc-mad()-Z | tBodyAcc-max()-X | ... | fBodyBodyGyroJerkMag-skewness() | fBodyB |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.288585 | -0.020294 | -0.132905 | -0.995279 | -0.983111 | -0.913526 | -0.995112 | -0.983185 | -0.923527 | -0.934724 | ... | -0.298676 | |
| **1** | 0.278419 | -0.016411 | -0.123520 | -0.998245 | -0.975300 | -0.960322 | -0.998807 | -0.974914 | -0.957686 | -0.943068 | ... | -0.595051 | |
| **2** | 0.279653 | -0.019467 | -0.113462 | -0.995380 | -0.967187 | -0.978944 | -0.996520 | -0.963668 | -0.977469 | -0.938692 | ... | -0.390748 | |
| **3** | 0.279174 | -0.026201 | -0.123283 | -0.996091 | -0.983403 | -0.990675 | -0.997099 | -0.982750 | -0.989302 | -0.938692 | ... | -0.117290 | |
| **4** | 0.276629 | -0.016570 | -0.115362 | -0.998139 | -0.980817 | -0.990482 | -0.998321 | -0.979672 | -0.990441 | -0.942469 | ... | -0.351471 | |

5 rows × 562 columns

In [3]:
```
# Display the shape of the DataFrame

df.shape  # 7352 - rows , 562 - Columns
```

Out[3]:
```
(7352, 562)
```

In [4]:
```
# Get the value counts of the 'Activity' column in the DataFrame

df['Activity'].value_counts()
```

Out[4]:
```
LAYING                1407
STANDING              1374
SITTING               1286
WALKING               1226
WALKING_UPSTAIRS      1073
WALKING_DOWNSTAIRS     986
Name: Activity, dtype: int64
```

## Duplicated Feature

In [5]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Separate features and target
X = df.drop('Activity', axis=1)
y = df['Activity']

# Encode target labels
le = LabelEncoder()
y = le.fit_transform(y)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [6]:
```
# Print the shape of X_train

print(X_train.shape)

# Print the shape of X_test
print(X_test.shape)
```

```
(5881, 561)
(1471, 561)
```

## Applying Logistic Regression

In [7]:
```
# Initialize and train logistic regression model
log_reg = LogisticRegression(max_iter=1000)  # Increase max_iter if it doesn't converge
log_reg.fit(X_train, y_train)


# Make predictions on the test set
y_pred = log_reg.predict(X_test)


# Calculate and print accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Test accuracy:", accuracy)
```

```
Test accuracy: 0.9809653297076818

C:\Users\user\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:444: ConvergenceWarning: lbfgs failed to converg
e (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/l
inear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

## 1. Removing Duplicate Columns

In [9]:
```
def get_duplicate_columns(df):
    """
    Returns a dictionary of duplicate columns in a DataFrame.

    Args:
        df (pandas.DataFrame): The input DataFrame.

    Returns:
        dict: A dictionary where the keys are duplicate columns and the values are lists of duplicate columns.
    """
    duplicate_columns = {}
    seen_columns = {}

    for column in df.columns:
        current_column = df[column]

        # Convert column data to bytes
        try:
            current_column_hash = current_column.values.tobytes()
        except AttributeError:
            current_column_hash = current_column.to_string().encode()

        if current_column_hash in seen_columns:
            if seen_columns[current_column_hash] in duplicate_columns:
                duplicate_columns[seen_columns[current_column_hash]].append(column)
            else:
                duplicate_columns[seen_columns[current_column_hash]] = [column]
        else:
            seen_columns[current_column_hash] = column

    return duplicate_columns
```

In [10]:
```
# Get duplicate columns in the dataset

duplicate_columns = get_duplicate_columns(X_train)
```

In [11]:
```
duplicate_columns

# Duplicated in key pair values
```

Out[11]:
```
{'tBodyAccMag-mean()': ['tBodyAccMag-sma()',
  'tGravityAccMag-mean()',
  'tGravityAccMag-sma()'],
 'tBodyAccMag-std()': ['tGravityAccMag-std()'],
 'tBodyAccMag-mad()': ['tGravityAccMag-mad()'],
 'tBodyAccMag-max()': ['tGravityAccMag-max()'],
 'tBodyAccMag-min()': ['tGravityAccMag-min()'],
 'tBodyAccMag-energy()': ['tGravityAccMag-energy()'],
 'tBodyAccMag-iqr()': ['tGravityAccMag-iqr()'],
 'tBodyAccMag-entropy()': ['tGravityAccMag-entropy()'],
 'tBodyAccMag-arCoeff()1': ['tGravityAccMag-arCoeff()1'],
 'tBodyAccMag-arCoeff()2': ['tGravityAccMag-arCoeff()2'],
 'tBodyAccMag-arCoeff()3': ['tGravityAccMag-arCoeff()3'],
 'tBodyAccMag-arCoeff()4': ['tGravityAccMag-arCoeff()4'],
 'tBodyAccJerkMag-mean()': ['tBodyAccJerkMag-sma()'],
 'tBodyGyroMag-mean()': ['tBodyGyroMag-sma()'],
 'tBodyGyroJerkMag-mean()': ['tBodyGyroJerkMag-sma()'],
 'fBodyAccMag-mean()': ['fBodyAccMag-sma()'],
 'fBodyBodyAccJerkMag-mean()': ['fBodyBodyAccJerkMag-sma()'],
 'fBodyBodyGyroMag-mean()': ['fBodyBodyGyroMag-sma()'],
 'fBodyBodyGyroJerkMag-mean()': ['fBodyBodyGyroJerkMag-sma()']}
```

In [12]:
```
X_train[['tBodyAccMag-mean()','tBodyAccMag-sma()','tGravityAccMag-mean()','tGravityAccMag-sma()']]
```

Out[12]:

| | tBodyAccMag-mean() | tBodyAccMag-sma() | tGravityAccMag-mean() | tGravityAccMag-sma() |
|---|---|---|---|---|
| **57** | -0.997420 | -0.997420 | -0.997420 | -0.997420 |
| **4154** | -0.133797 | -0.133797 | -0.133797 | -0.133797 |
| **6945** | 0.130528 | 0.130528 | 0.130528 | 0.130528 |
| **527** | -0.955222 | -0.955222 | -0.955222 | -0.955222 |
| **4196** | -0.992499 | -0.992499 | -0.992499 | -0.992499 |
| **...** | ... | ... | ... | ... |
| **5191** | -0.372758 | -0.372758 | -0.372758 | -0.372758 |
| **5226** | -0.256033 | -0.256033 | -0.256033 | -0.256033 |
| **5390** | -0.434138 | -0.434138 | -0.434138 | -0.434138 |
| **860** | -0.983969 | -0.983969 | -0.983969 | -0.983969 |
| **7270** | -0.200163 | -0.200163 | -0.200163 | -0.200163 |

5881 rows × 4 columns

In [13]:
```
# Remove duplicate columns from X_train and X_test

for one_list in duplicate_columns.values():
    X_train.drop(columns=one_list,inplace=True)
    X_test.drop(columns=one_list,inplace=True)
```

In [14]:
```
print(X_train.shape)
```

```
(5881, 540)
```

In [15]:
```
print(X_test.shape)
```

```
(1471, 540)
```

## 2. Variance Threshold

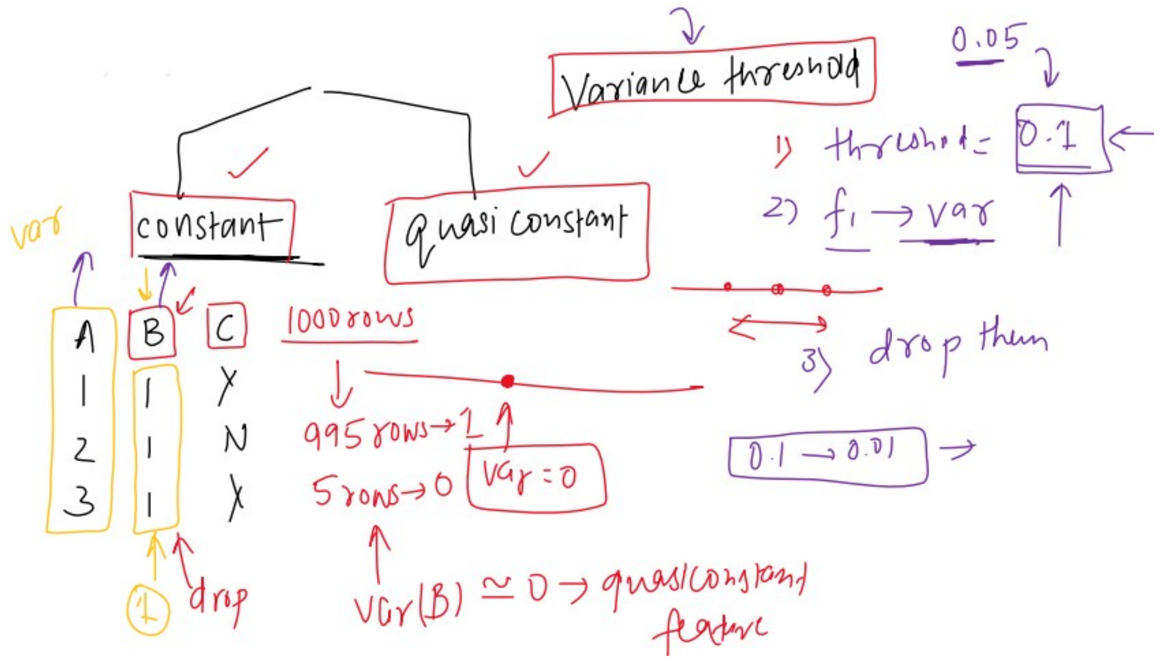
Variance thresholding is a simple filter-based feature selection method that selects features based on their variance. The idea behind this method is that features with low variance carry less information and may be less informative for predictive modeling tasks. Therefore, by setting a threshold for the variance, features with variance below that threshold are removed from consideration.

- **Constant Features**: Constant features are those that have the same value for all data points in the dataset. Since these features do not change or vary across the dataset, they carry no useful information for modeling or analysis. Examples of constant features include an ID column, a feature that was mistakenly duplicated, or a feature that was created with incorrect data.

Removing constant features is typically straightforward and can be done by calculating the variance of each feature. If the variance is zero, it indicates that the feature is constant and can be safely eliminated from the dataset.

- **Quasi-constant Features**: Quasi-constant features, also known as near-constant or low-variance features, are those that have very low variability with values that are almost constant. Although these features exhibit some degree of variation, their low variance suggests that they do not provide much discriminatory or informative power.

Determining the threshold to classify a feature as quasi-constant depends on the specific dataset and problem. It is common to set a threshold based on empirical analysis or domain knowledge.



```
In [16]: # Code


         from sklearn.feature_selection import VarianceThreshold


         # Create a VarianceThreshold object with a threshold of 0.05
         sel = VarianceThreshold(threshold=0.05)
```

```
In [17]: # Fit the model to the training data

         sel.fit(X_train)
```

```
Out[17]: ▾       VarianceThreshold
         VarianceThreshold(threshold=0.05)
```

```
In [18]: # Get the support of the current selection


         # sel.get_support()

         """ get_support = It will return an array of boolean values. True for the features whose importance is greater than
         the mean importance and False for the rest."""
```

```
Out[18]: ' get_support = It will return an array of boolean values. True for the features whose importance is greater than \nthe mean
         importance and False for the rest.'
```

```
array([False, False, False,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True, False,
       False,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True, False, False, False,  True, False,  True,
        True,  True,  True,  True,  True,  True,  True, False, False,
       False, False, False, False,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True, False, False, False,  True,
        True,  True, False, False, False, False,  True,  True,  True,
```

```
In [19]: # Calculates the sum of the selected support.

         sum(sel.get_support()) # 349 are greater than Threshold
```

```
Out[19]: 349
```

In [20]:
```
# Retrieve the selected columns from X_train

columns = X_train.columns[sel.get_support()]
```

In [21]:
```
columns

# columns are greater than Threshold
```

Out[21]:
```
Index(['tBodyAcc-std()-X', 'tBodyAcc-std()-Y', 'tBodyAcc-std()-Z',
       'tBodyAcc-mad()-X', 'tBodyAcc-mad()-Y', 'tBodyAcc-mad()-Z',
       'tBodyAcc-max()-X', 'tBodyAcc-max()-Y', 'tBodyAcc-max()-Z',
       'tBodyAcc-min()-X',
       ...
       'fBodyBodyGyroJerkMag-meanFreq()', 'fBodyBodyGyroJerkMag-skewness()',
       'fBodyBodyGyroJerkMag-kurtosis()', 'angle(tBodyAccMean,gravity)',
       'angle(tBodyAccJerkMean),gravityMean)',
       'angle(tBodyGyroMean,gravityMean)',
       'angle(tBodyGyroJerkMean,gravityMean)', 'angle(X,gravityMean)',
       'angle(Y,gravityMean)', 'angle(Z,gravityMean)'],
      dtype='object', length=349)
```

In [22]:
```

# Transform the features of the training set using sel ( Keep OnLy 349 columns and delete every column , where below threshol

X_train = sel.transform(X_train)

# Transform the features of the testing set using sel

X_test = sel.transform(X_test)


# Create DataFrames for the transformed training and test sets with the specified column names
#because they conveted in to numpy arrays

X_train = pd.DataFrame(X_train, columns=columns)
X_test = pd.DataFrame(X_test, columns=columns)
```

In [23]:
```
print(X_train.shape)

print(X_test.shape)

# Here We got 349 Columns , Firstly data consists of 540 columns
```

```
(5881, 349)
(1471, 349)
```

In [24]:
```
# Print the first few rows of the X_train dataframe

X_train.head()
```

Out[24]:

| | tBodyAcc-std()-X | tBodyAcc-std()-Y | tBodyAcc-std()-Z | tBodyAcc-mad()-X | tBodyAcc-mad()-Y | tBodyAcc-mad()-Z | tBodyAcc-max()-X | tBodyAcc-max()-Y | tBodyAcc-max()-Z | tBodyAcc-min()-X | ... | fBodyBodyGyroJerkMag-meanFreq() | fBodyB |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -0.994425 | -0.994873 | -0.994886 | -0.994939 | -0.993994 | -0.995450 | -0.938974 | -0.577031 | -0.813863 | 0.846922 | ... | 0.394506 | |
| 1 | -0.326331 | 0.069663 | -0.224321 | -0.343326 | 0.039623 | -0.256327 | -0.310961 | 0.085617 | -0.411806 | 0.271334 | ... | 0.052089 | |
| 2 | -0.026220 | -0.032163 | 0.393109 | -0.118256 | -0.030279 | 0.432861 | 0.370607 | -0.072309 | 0.200747 | 0.118277 | ... | -0.038923 | |
| 3 | -0.981092 | -0.901124 | -0.960423 | -0.984417 | -0.901405 | -0.965788 | -0.922291 | -0.524676 | -0.807362 | 0.825370 | ... | -0.145084 | |
| 4 | -0.997380 | -0.983893 | -0.984482 | -0.997331 | -0.985196 | -0.983768 | -0.942062 | -0.564033 | -0.810993 | 0.853330 | ... | 0.096524 | |

5 rows × 349 columns

**Points to Consider**

1. **Ignores Target Variable**: Variance Threshold is a univariate method, meaning it evaluates each feature independently and doesn't consider the relationship between each feature and the target variable. This means it may keep irrelevant features that have a high variance but no relationship with the target, or discard potentially useful features that have a low variance but a strong relationship with the target.

2. **Ignores Feature Interactions**: Variance Threshold doesn't account for interactions between features. A feature with a low variance may become very informative when combined with another feature.

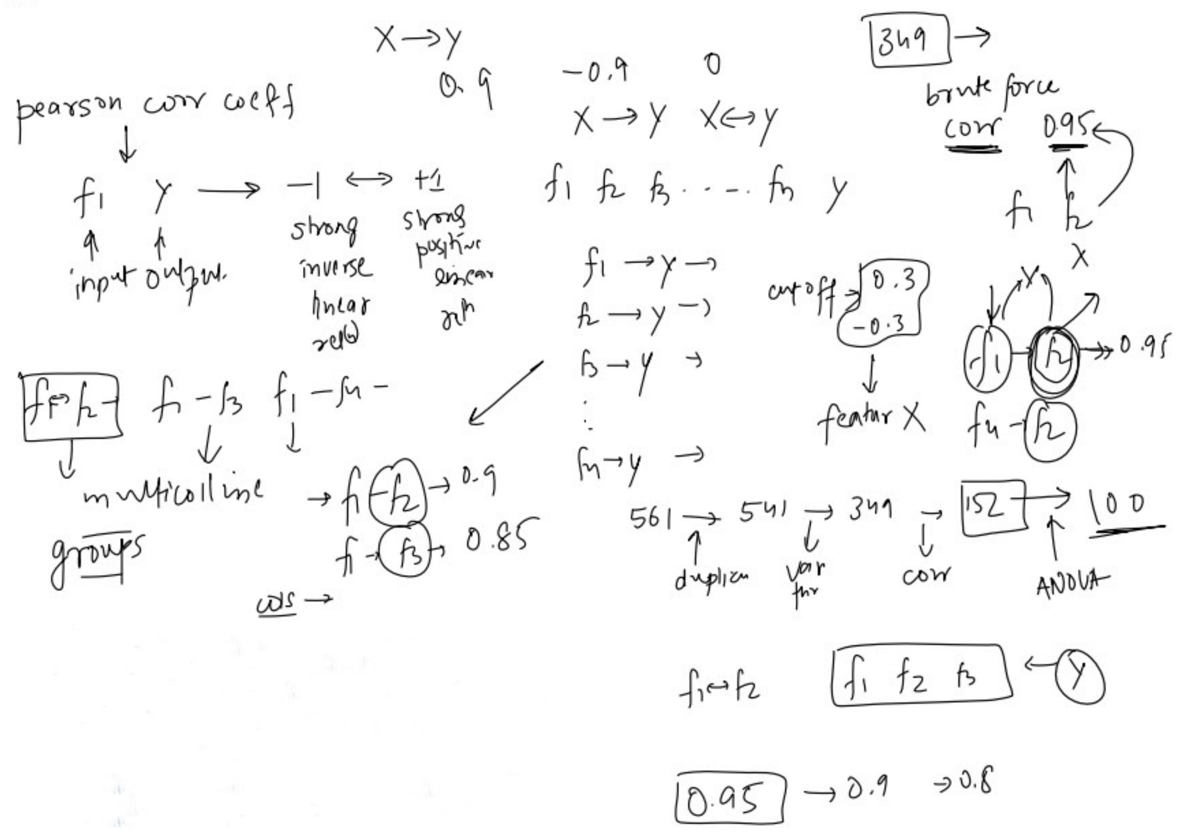
3. **Sensitive to Data Scaling**: Variance Threshold is sensitive to the scale of the data. If features are not on the same scale, the variance will naturally

4. **Arbitrary Threshold Value**: It's up to the user to define what constitutes a "low" variance. The threshold is not always easy to define and the optimal value can vary between datasets.

### 3. Correlation

Correlation-based Feature Selection: This method measures the correlation between each feature and the target variable. Features with higher correlation values are considered more relevant and are selected for inclusion. Common correlation measures include Pearson correlation coefficient for continuous variables and point biserial correlation for categorical variables.

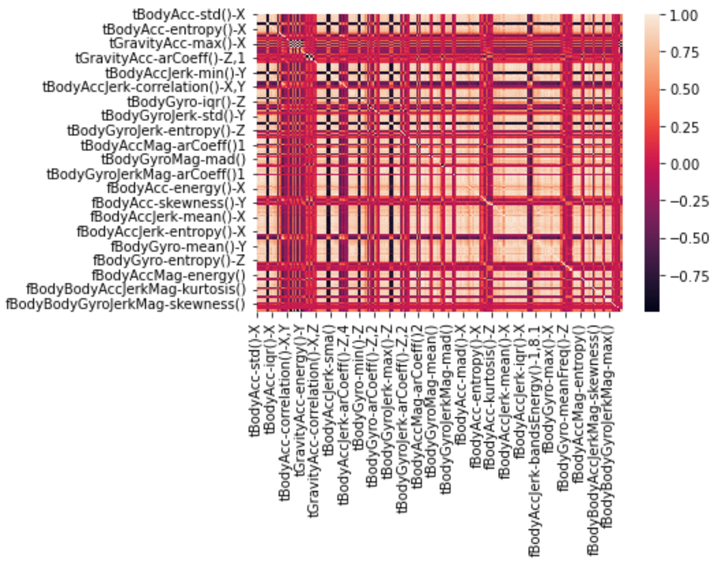
### Explanation :



```
In [26]: # code

         import seaborn as sns
```

In [27]:
```python
# Create a heatmap of the correlation matrix of X_train

sns.heatmap(X_train.corr())
```

Out[27]:
```
<AxesSubplot:>
```



In [29]:
```python
X_train.corr()
```

Out[29]:

| | tBodyAcc-std()-X | tBodyAcc-std()-Y | tBodyAcc-std()-Z | tBodyAcc-mad()-X | tBodyAcc-mad()-Y | tBodyAcc-mad()-Z | tBodyAcc-max()-X | tBodyAcc-max()-Y | tBodyAcc-max()-Z | tBodyAcc-min()-X |
|---|---|---|---|---|---|---|---|---|---|---|
| **tBodyAcc-std()-X** | 1.000000 | 0.927247 | 0.850268 | 0.998631 | 0.920936 | 0.845200 | 0.981284 | 0.893743 | 0.843918 | -0.966714 |
| **tBodyAcc-std()-Y** | 0.927247 | 1.000000 | 0.895065 | 0.922627 | 0.997384 | 0.894128 | 0.917831 | 0.953852 | 0.882782 | -0.937472 |
| **tBodyAcc-std()-Z** | 0.850268 | 0.895065 | 1.000000 | 0.842986 | 0.890973 | 0.997414 | 0.852711 | 0.864716 | 0.936311 | -0.861033 |
| **tBodyAcc-mad()-X** | 0.998631 | 0.922627 | 0.842986 | 1.000000 | 0.916201 | 0.838010 | 0.973704 | 0.888702 | 0.838024 | -0.962447 |
| **tBodyAcc-mad()-Y** | 0.920936 | 0.997384 | 0.890973 | 0.916201 | 1.000000 | 0.890707 | 0.911283 | 0.950131 | 0.877793 | -0.932521 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **angle(tBodyGyroMean,gravityMean)** | 0.023914 | -0.002241 | -0.010535 | 0.024098 | -0.005865 | -0.014838 | 0.029230 | -0.000207 | -0.023622 | -0.008700 |
| **angle(tBodyGyroJerkMean,gravityMean)** | -0.035176 | -0.028881 | -0.016002 | -0.035629 | -0.026679 | -0.016949 | -0.038935 | -0.013144 | -0.011510 | 0.030630 |
| **angle(X,gravityMean)** | -0.374114 | -0.383095 | -0.344114 | -0.370629 | -0.379578 | -0.346350 | -0.386159 | -0.373556 | -0.345776 | 0.365571 |
| **angle(Y,gravityMean)** | 0.472605 | 0.524945 | 0.475241 | 0.467965 | 0.526803 | 0.476498 | 0.482312 | 0.489971 | 0.462052 | -0.471464 |
| **angle(Z,gravityMean)** | 0.393209 | 0.432180 | 0.480824 | 0.389139 | 0.430548 | 0.477627 | 0.404088 | 0.424181 | 0.416249 | -0.392253 |

349 rows × 349 columns

In [30]:
```python
corr_matrix = X_train.corr()
```

In [31]:
```python
# Get the column names of the DataFrame
columns = corr_matrix.columns

# Create an empty list to keep track of columns to drop
columns_to_drop = []

# Loop over the columns
for i in range(len(columns)):
    for j in range(i + 1, len(columns)):

        # Access the cell of the DataFrame
        if corr_matrix.loc[columns[i], columns[j]] > 0.95: # if corelation is greater than 0.95 drop them, and add remaining
            columns_to_drop.append(columns[j])

print(len(columns_to_drop))
```

```
1508
```

In [33]:
```
set(columns_to_drop)

"""The line set(columns_to_drop) creates a set from the columns_to_drop list.
A set is an unordered collection of unique elements.
In this case, it creates a set of column names that need to be dropped based on the correlation condition.

"""
```

Out[33]:
```
{'fBodyAcc-bandsEnergy()-1,16',
 'fBodyAcc-bandsEnergy()-1,16.1',
 'fBodyAcc-bandsEnergy()-1,16.2',
 'fBodyAcc-bandsEnergy()-1,24',
 'fBodyAcc-bandsEnergy()-1,24.1',
 'fBodyAcc-bandsEnergy()-1,24.2',
 'fBodyAcc-bandsEnergy()-1,8',
 'fBodyAcc-bandsEnergy()-17,32',
 'fBodyAcc-energy()-X',
 'fBodyAcc-energy()-Y',
 'fBodyAcc-entropy()-X',
 'fBodyAcc-entropy()-Y',
 'fBodyAcc-entropy()-Z',
 'fBodyAcc-iqr()-X',
 'fBodyAcc-iqr()-Y',
 'fBodyAcc-iqr()-Z',
 'fBodyAcc-kurtosis()-X',
 'fBodyAcc-kurtosis()-Y',
 'fBodyAcc-kurtosis()-Z',
```

In [34]:
```
columns_to_drop = set(columns_to_drop)
```

In [35]:
```
# The code simply calculates the length of the list 'columns_to_drop'.


len(columns_to_drop)

# By using Correlation , we have dropped from 349 columns to 197 columns , we got total 152 columns remaining
```

Out[35]:
```
197
```

In [36]:
```
# Remove specified columns from X_train and X_test

X_train.drop(columns = columns_to_drop, axis = 1, inplace=True)

X_test.drop(columns = columns_to_drop, axis = 1, inplace=True)
```

In [39]:
```
# Print the shape of the training and testing data

print(X_train.shape)
print(X_test.shape)
```

```
(5881, 152)
(1471, 152)
```

In [41]:
```
X_train
```

Out[41]:

| | tBodyAcc-std()-X | tBodyAcc-std()-Y | tBodyAcc-std()-Z | tBodyAcc-max()-Z | tBodyAcc-min()-X | tBodyAcc-min()-Y | tBodyAcc-min()-Z | tBodyAcc-entropy()-X | tBodyAcc-entropy()-Y | tBodyAcc-entropy()-Z | ... | fBodyBodyGyroMag-skewness() | fBodyBo |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | -0.994425 | -0.994873 | -0.994886 | -0.813863 | 0.846922 | 0.691468 | 0.846423 | -0.611174 | -0.768785 | -0.663066 | ... | 0.043182 | |
| **1** | -0.326331 | 0.069663 | -0.224321 | -0.411806 | 0.271334 | 0.039452 | 0.269204 | 0.403663 | 0.180054 | 0.176069 | ... | -0.225796 | |
| **2** | -0.026220 | -0.032163 | 0.393109 | 0.200747 | 0.118277 | 0.072295 | 0.245986 | 0.318557 | 0.135103 | 0.087680 | ... | -0.346388 | |
| **3** | -0.981092 | -0.901124 | -0.960423 | -0.807362 | 0.825370 | 0.642789 | 0.815368 | -0.376515 | -0.171730 | -0.496816 | ... | 0.197311 | |
| **4** | -0.997380 | -0.983893 | -0.984482 | -0.810993 | 0.853330 | 0.687431 | 0.844895 | -0.652548 | -0.678458 | -0.486837 | ... | 0.310607 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **5876** | -0.555352 | -0.104055 | -0.438064 | -0.373579 | 0.454289 | 0.153427 | 0.475699 | 0.455070 | 0.288106 | 0.191017 | ... | -0.490616 | |
| **5877** | -0.290043 | -0.212102 | -0.469731 | -0.123285 | 0.123907 | 0.202561 | 0.577615 | 0.345742 | 0.159941 | 0.085557 | ... | -0.846175 | |
| **5878** | -0.627198 | -0.216566 | -0.424764 | -0.441355 | 0.530629 | 0.077330 | 0.469639 | 0.436434 | 0.322259 | 0.086872 | ... | -0.609038 | |
| **5879** | -0.994825 | -0.985314 | -0.965857 | -0.805684 | 0.849776 | 0.688222 | 0.828575 | -0.542681 | -0.667647 | -0.431412 | ... | -0.327943 | |
| **5880** | -0.368655 | -0.142631 | -0.151250 | -0.108235 | 0.252673 | 0.303901 | 0.326319 | 0.407481 | 0.281442 | 0.105444 | ... | 0.126431 | |

5881 rows × 152 columns

**Disadvantages of Correlation**

1. **Linearity Assumption**: Correlation measures the linear relationship between two variables. It does not capture non-linear relationships well. If a relationship is nonlinear, the correlation coefficient can be misleading.

2. **Doesn't Capture Complex Relationships**: Correlation only measures the relationship between two variables at a time. It may not capture complex relationships involving more than two variables.

3. **Threshold Determination**: Just like variance threshold, defining what level of correlation is considered "high" can be subjective and may vary depending on the specific problem or dataset.

4. **Sensitive to Outliers**: Correlation is sensitive to outliers. A few extreme values can significantly skew the correlation coefficient.

## 4. ANOVA

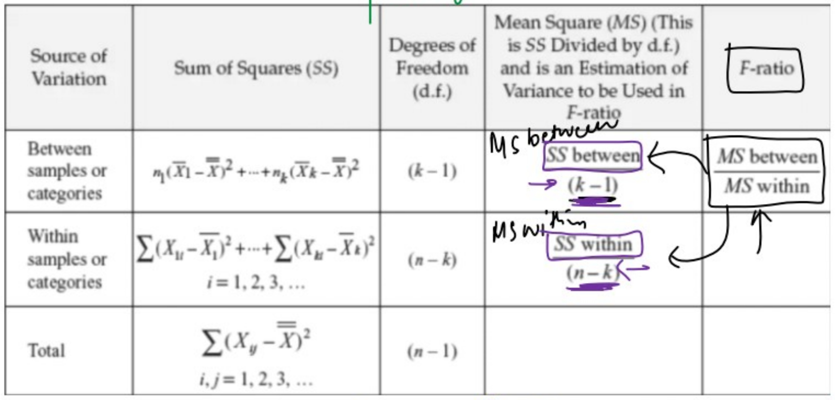
ANOVA can be used as a technique to assess the relevance of categorical features with respect to a continuous target variable. By comparing the means of the target variable across different levels or groups of the categorical feature, ANOVA helps determine if there are significant differences in the means, indicating the potential predictive power of the feature

**Explanation :**

$\boxed{f_1 \longrightarrow y}$ rel$^{\text{n}}$ $k-1$ $n \to$ #rows $k \to$ # category 6

↑ strong $n-k$

→ Hypothesis test → 1 way ANOVA



| Source of Variation | Sum of Squares ($SS$) | Degrees of Freedom (d.f.) | Mean Square ($MS$) (This is $SS$ Divided by d.f.) and is an Estimation of Variance to be Used in $F$-ratio | $\boxed{F\text{-ratio}}$ |
|---|---|---|---|---|
| Between samples or categories | $n_1(\overline{X}_1-\overline{\overline{X}})^2+\cdots+n_k(\overline{X}_k-\overline{\overline{X}})^2$ | $(k-1)$ | MS between $\dfrac{\boxed{SS\text{ between}}}{(k-1)}$ | $\boxed{\dfrac{MS\text{ between}}{MS\text{ within}}}$ |
| Within samples or categories | $\sum(X_{1i}-\overline{X}_1)^2+\cdots+\sum(X_{ki}-\overline{X}_k)^2$ $i=1,2,3,\ldots$ | $(n-k)$ | MS within $\dfrac{\boxed{SS\text{ within}}}{(n-k)}$ | |
| Total | $\sum(X_{ij}-\overline{\overline{X}})^2$ $i,j=1,2,3,\ldots$ | $(n-1)$ | | |

numerical $f_1$ → $y$ categorical ↓ $>2$ classes

↓

numerical $f_1$ → $y$ numerical

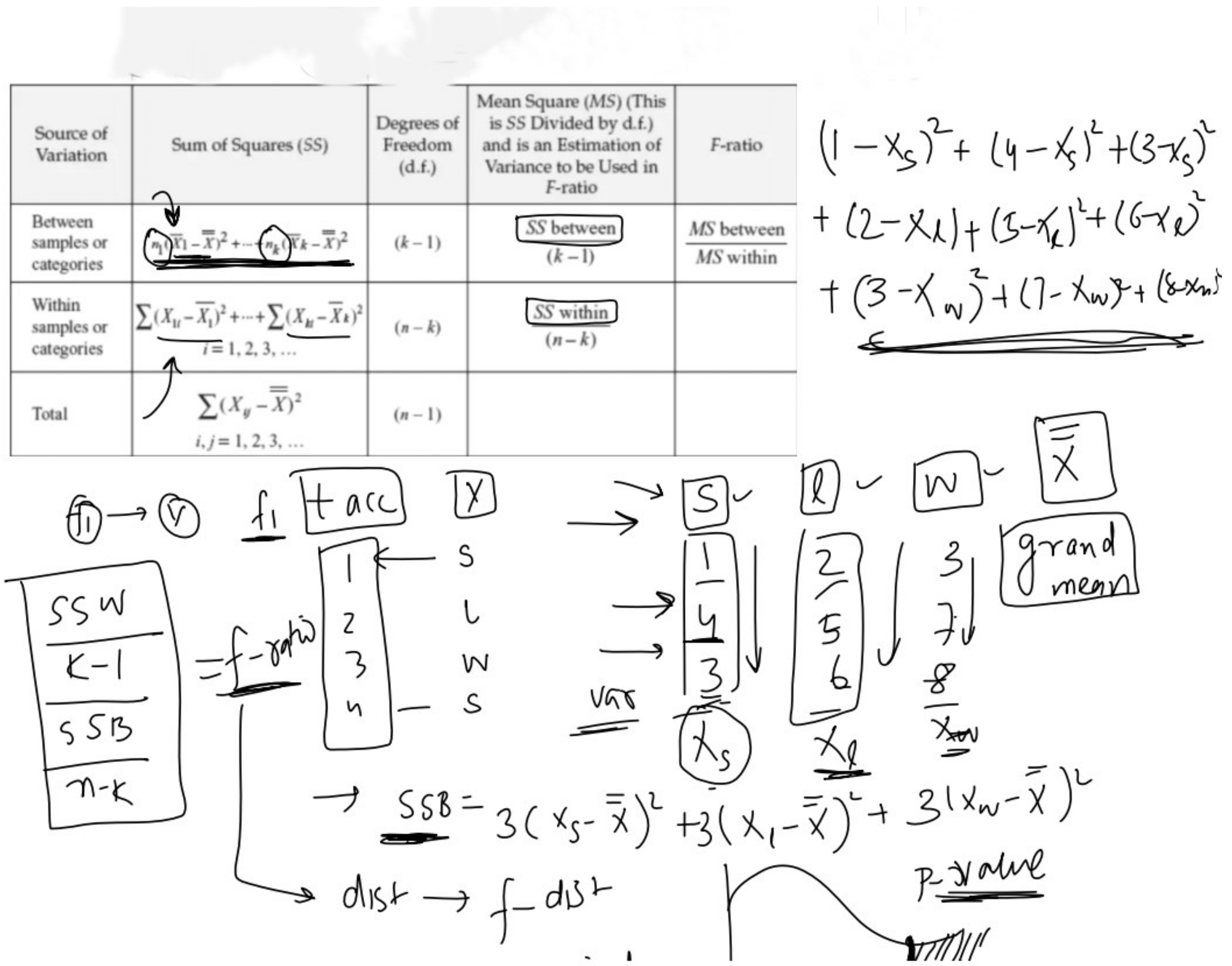
f-statistic → p-value → rel$^{\text{n}}$ $\overrightarrow{f \to y}$

$\boxed{f_1\ f_2 \ldots\ f_{152}}$ $y$ → cat > 2, ← num

ANOVA

→ $f_1$ → $y$ → no rel$^{\text{n}}$ → sklearn

SSW = $\boxed{150}$



| Source of Variation | Sum of Squares (SS) | Degrees of Freedom (d.f.) | Mean Square (MS) (This is SS Divided by d.f.) and is an Estimation of Variance to be Used in F-ratio | F-ratio |
|---|---|---|---|---|
| Between samples or categories | $n_1(\bar{X}_1-\bar{\bar{X}})^2+\cdots+n_k(\bar{X}_k-\bar{\bar{X}})^2$ | $(k-1)$ | $\frac{\boxed{SS\text{ between}}}{(k-1)}$ | $\frac{MS\text{ between}}{MS\text{ within}}$ |
| Within samples or categories | $\sum(X_{1i}-\bar{X}_1)^2+\cdots+\sum(X_{ki}-\bar{X}_k)^2$ $i=1,2,3,\ldots$ | $(n-k)$ | $\frac{\boxed{SS\text{ within}}}{(n-k)}$ | |
| Total | $\sum(X_{ij}-\bar{\bar{X}})^2$ $i,j=1,2,3,\ldots$ | $(n-1)$ | | |

$(1-\bar{X}_s)^2+(4-\bar{X}_s)^2+(3-\bar{X}_s)^2$
$+(2-\bar{X}_l)+(5-\bar{X}_l)^2+(6-\bar{X}_l)^2$
$+(3-\bar{X}_w)^2+(7-\bar{X}_w)^2+(8-\bar{X}_w)$

$f_1 \rightarrow$ V  $f_1$ $\boxed{t\ acc}$

| $\boxed{X}$ | |
|---|---|
| 1 | s |
| 2 | l |
| 3 | w |
| 4 | s |

| $\boxed{S}$ | $\boxed{l}$ | $\boxed{w}$ |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 7 |
| 3 | 6 | 8 |
| $\bar{X}_s$ | $\bar{X}_l$ | $\bar{X}_w$ |

var

$\boxed{\bar{\bar{X}}}$ $\boxed{\text{grand mean}}$

$$\boxed{\frac{\frac{SSW}{K-1}}{\frac{SSB}{n-K}}} = f\text{-ratio}$$

$\rightarrow SSB = 3(\bar{x}_s-\bar{\bar{x}})^2+3(\bar{x}_l-\bar{\bar{x}})^2+3(\bar{x}_w-\bar{\bar{x}})^2$

$\rightarrow$ dist $\rightarrow$ f-dist

p-value

In [42]:
```
# code

from sklearn.feature_selection import f_classif # Anova

from sklearn.feature_selection import SelectKBest # Select ' K' Best Features

# Perform feature selection

sel = SelectKBest(f_classif, k=100).fit(X_train, y_train)

# display selected feature names
X_train.columns[sel.get_support()]


# These are 100 best columns
```

Out[42]:
```
Index(['tBodyAcc-std()-X', 'tBodyAcc-std()-Y', 'tBodyAcc-std()-Z',
       'tBodyAcc-max()-Z', 'tBodyAcc-min()-X', 'tBodyAcc-min()-Y',
       'tBodyAcc-min()-Z', 'tBodyAcc-entropy()-X', 'tBodyAcc-entropy()-Y',
       'tBodyAcc-entropy()-Z', 'tBodyAcc-arCoeff()-X,1',
       'tBodyAcc-arCoeff()-X,2', 'tBodyAcc-arCoeff()-X,3',
       'tBodyAcc-arCoeff()-Y,1', 'tBodyAcc-arCoeff()-Z,1',
       'tBodyAcc-correlation()-X,Y', 'tBodyAcc-correlation()-Y,Z',
       'tGravityAcc-mean()-X', 'tGravityAcc-mean()-Y', 'tGravityAcc-mean()-Z',
       'tGravityAcc-sma()', 'tGravityAcc-energy()-Y', 'tGravityAcc-energy()-Z',
       'tGravityAcc-entropy()-X', 'tGravityAcc-entropy()-Y',
       'tGravityAcc-arCoeff()-Y,1', 'tGravityAcc-arCoeff()-Y,2',
       'tGravityAcc-arCoeff()-Z,1', 'tGravityAcc-arCoeff()-Z,2',
       'tGravityAcc-correlation()-Y,Z', 'tBodyAccJerk-std()-Z',
       'tBodyAccJerk-min()-X', 'tBodyAccJerk-min()-Y', 'tBodyAccJerk-min()-Z',
       'tBodyAccJerk-entropy()-X', 'tBodyAccJerk-arCoeff()-X,3',
       'tBodyAccJerk-correlation()-X,Y', 'tBodyGyro-std()-X',
       'tBodyGyro-std()-Y', 'tBodyGyro-std()-Z', 'tBodyGyro-max()-X',
       'tBodyGyro-max()-Z', 'tBodyGyro-min()-X', 'tBodyGyro-min()-Z',
       'tBodyGyro-entropy()-X', 'tBodyGyro-entropy()-Y',
       'tBodyGyro-entropy()-Z', 'tBodyGyro-arCoeff()-X,1',
       'tBodyGyro-arCoeff()-X,2', 'tBodyGyro-arCoeff()-Z,1',
       'tBodyGyro-arCoeff()-Z,2', 'tBodyGyro-arCoeff()-Z,3',
       'tBodyGyro-correlation()-Y,Z', 'tBodyGyroJerk-std()-Y',
       'tBodyGyroJerk-min()-X', 'tBodyGyroJerk-min()-Y',
       'tBodyGyroJerk-min()-Z', 'tBodyGyroJerk-arCoeff()-Z,2',
       'tBodyGyroJerk-arCoeff()-Z,3', 'tBodyAccMag-arCoeff()1',
       'tBodyAccMag-arCoeff()2', 'tBodyAccMag-arCoeff()3',
       'tBodyAccJerkMag-min()', 'tBodyAccJerkMag-arCoeff()1',
       'tBodyAccJerkMag-arCoeff()2', 'tBodyGyroMag-min()',
       'tBodyGyroMag-energy()', 'tBodyGyroMag-entropy()',
       'tBodyGyroJerkMag-min()', 'fBodyAcc-min()-X', 'fBodyAcc-energy()-Z',
       'fBodyAcc-meanFreq()-X', 'fBodyAcc-meanFreq()-Y',
       'fBodyAcc-meanFreq()-Z', 'fBodyAcc-skewness()-X',
       'fBodyAcc-bandsEnergy()-17,24', 'fBodyAcc-bandsEnergy()-1,8.1',
       'fBodyAcc-bandsEnergy()-9,16.1', 'fBodyAcc-bandsEnergy()-17,32.1',
       'fBodyAcc-bandsEnergy()-1,8.2', 'fBodyAccJerk-maxInds-X',
       'fBodyAccJerk-maxInds-Z', 'fBodyAccJerk-meanFreq()-X',
       'fBodyAccJerk-meanFreq()-Y', 'fBodyAccJerk-meanFreq()-Z',
       'fBodyAccJerk-skewness()-X', 'fBodyAccJerk-bandsEnergy()-1,8.1',
       'fBodyGyro-maxInds-Y', 'fBodyGyro-maxInds-Z', 'fBodyGyro-meanFreq()-X',
       'fBodyGyro-meanFreq()-Z', 'fBodyGyro-bandsEnergy()-1,8.1',
       'fBodyAccMag-maxInds', 'fBodyBodyAccJerkMag-min()',
       'fBodyBodyAccJerkMag-meanFreq()', 'fBodyBodyAccJerkMag-skewness()',
       'fBodyBodyGyroMag-meanFreq()', 'angle(X,gravityMean)',
       'angle(Y,gravityMean)', 'angle(Z,gravityMean)'],
      dtype='object')
```

In [43]:
```

# stored in columns

columns = X_train.columns[sel.get_support()]
```

In [44]:
```

# Transform X_train and X_test using sel
X_train = sel.transform(X_train)
X_test = sel.transform(X_test)

# Convert the transformed arrays into DataFrames with column names
X_train = pd.DataFrame(X_train, columns=columns)
X_test = pd.DataFrame(X_test, columns=columns)
```

In [45]:
```python
print(X_train.shape)
print(X_test.shape)

# from 152 column , we have got 100 Columns using Anova Method
```

```
(5881, 100)
(1471, 100)
```

In [46]:
```python
X_train.head()
```

Out[46]:

| | tBodyAcc-std()-X | tBodyAcc-std()-Y | tBodyAcc-std()-Z | tBodyAcc-max()-Z | tBodyAcc-min()-X | tBodyAcc-min()-Y | tBodyAcc-min()-Z | tBodyAcc-entropy()-X | tBodyAcc-entropy()-Y | tBodyAcc-entropy()-Z | ... | fBodyGyro-meanFreq()-Z | fBodyGyro-bandsEnergy()-1,8.1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | -0.994425 | -0.994873 | -0.994886 | -0.813863 | 0.846922 | 0.691468 | 0.846423 | -0.611174 | -0.768785 | -0.663066 | ... | -0.158283 | -0.999962 |
| **1** | -0.326331 | 0.069663 | -0.224321 | -0.411806 | 0.271334 | 0.039452 | 0.269204 | 0.403663 | 0.180054 | 0.176069 | ... | -0.315738 | -0.898022 |
| **2** | -0.026220 | -0.032163 | 0.393109 | 0.200747 | 0.118277 | 0.072295 | 0.245986 | 0.318557 | 0.135103 | 0.087680 | ... | -0.071045 | 0.192158 |
| **3** | -0.981092 | -0.901124 | -0.960423 | -0.807362 | 0.825370 | 0.642789 | 0.815368 | -0.376515 | -0.171730 | -0.496816 | ... | -0.629423 | -0.996263 |
| **4** | -0.997380 | -0.983893 | -0.984482 | -0.810993 | 0.853330 | 0.687431 | 0.844895 | -0.652548 | -0.678458 | -0.486837 | ... | 0.261846 | -0.999927 |

5 rows × 100 columns

**Disadvantages of Anova**:

1. **Assumption of Normality**: ANOVA assumes that the data for each group follow a normal distribution. This assumption may not hold true for all datasets, especially those with skewed distributions.

2. **Assumption of Homogeneity of Variance**: ANOVA assumes that the variances of the different groups are equal. This is the assumption of homogeneity of variance (also known as homoscedasticity). If this assumption is violated, it may lead to incorrect results.

3. **Independence of Observations**: ANOVA assumes that the observations are independent of each other. This might not be the case in datasets where observations are related (e.g., time series data, nested data).

4. **Effect of Outliers**: ANOVA is sensitive to outliers. A single outlier can significantly affect the F-statistic leading to a potentially erroneous conclusion.

5. **Doesn't Account for Interactions**: Just like other univariate feature selection methods, ANOVA does not consider interactions between features.

**Moment of Truth**

In [51]:
```python
# Initialize and train logistic regression model
log_reg = LogisticRegression(max_iter=1000)  # Increase max_iter if it doesn't converge
log_reg.fit(X_train, y_train)

# Make predictions on the test set
y_pred = log_reg.predict(X_test)

# Calculate and print accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Test accuracy:", accuracy)
```

```
Test accuracy: 0.9700883752549286
```

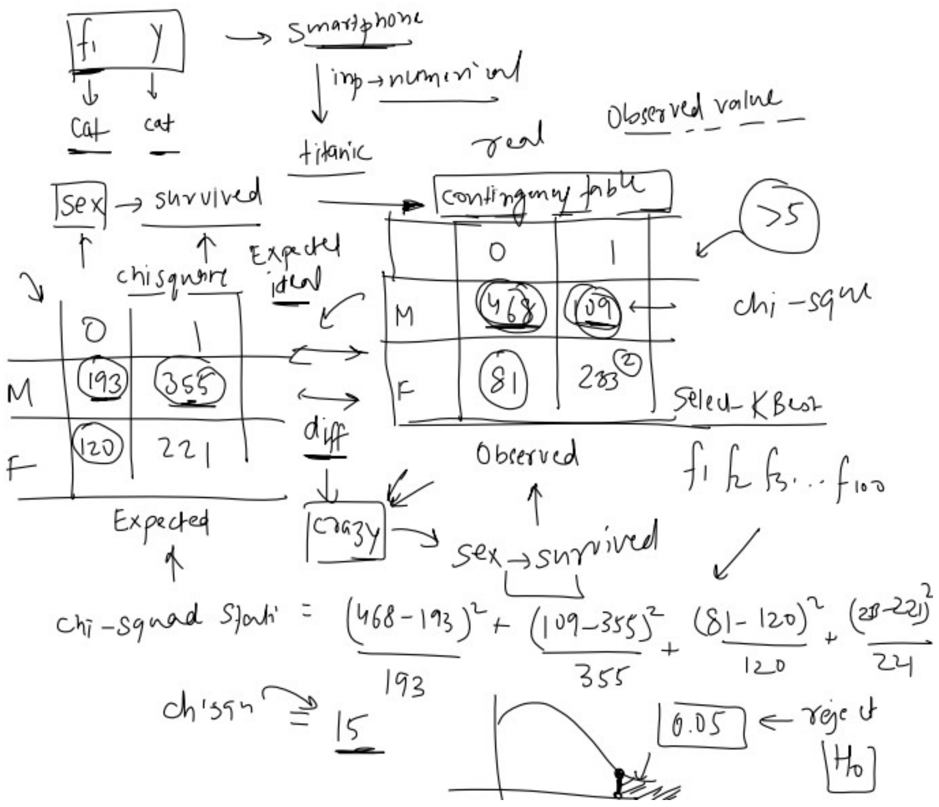
**We achieved the same accuracy even after reducing the number of columns from 562 to 100**

## 5.Chi-Square

Chi-square is a statistical test commonly used in feature selection to determine the independence between categorical variables. It assesses the relationship between two variables by comparing the observed frequencies in a contingency table with the frequencies that would be expected if the variables were independent.

- In the context of feature selection, the chi-square test can be used to evaluate the significance of the association between each feature and the target variable. The basic idea is to calculate the chi-square statistic for each feature and the target, and then assess the statistical significance of the relationship.

**Explaantion :**



In [54]:
```
titanic = pd.read_csv("D:\\datascience\\Nitish sir\\Inferential Statistics\\titanic_train.csv")[['Pclass','Sex','SibSp','Parcl
titanic.head()
```

Out[54]:

|   | Pclass | Sex | SibSp | Parch | Embarked | Survived |
|---|---|---|---|---|---|---|
| **0** | 3 | male | 1 | 0 | S | 0 |
| **1** | 1 | female | 1 | 0 | C | 1 |
| **2** | 3 | female | 0 | 0 | S | 1 |
| **3** | 1 | female | 1 | 0 | S | 1 |
| **4** | 3 | male | 0 | 0 | S | 0 |

In [55]:
```
ct = pd.crosstab(titanic['Survived'],titanic['Sex'],margins=True)
ct
```

Out[55]:

| Sex | female | male | All |
|---|---|---|---|
| **Survived** | | | |
| **0** | 81 | 468 | 549 |
| **1** | 233 | 109 | 342 |
| **All** | 314 | 577 | 891 |

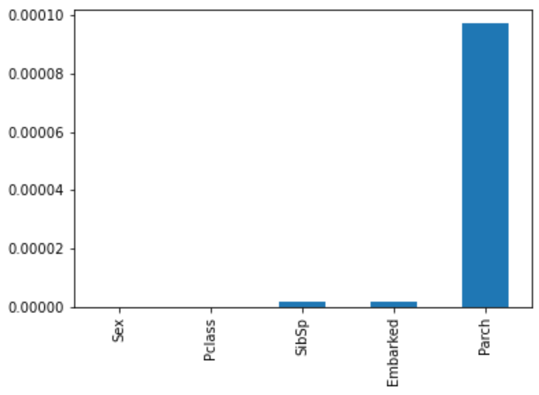
In [56]:
```
from scipy.stats import chi2_contingency

# Run the chi-squared test for independence

chi2_contingency(ct) # On SEX data

# Here are Expected Values with 'P value'

# Smaller the area , Greater the impact in data
```

Out[56]:
```
(263.05057407065567,
 1.0036732821369117e-55,
 4,
 array([[193.47474747, 355.52525253, 549.        ],
        [120.52525253, 221.47474747, 342.        ],
        [314.        , 577.        , 891.        ]]))
```

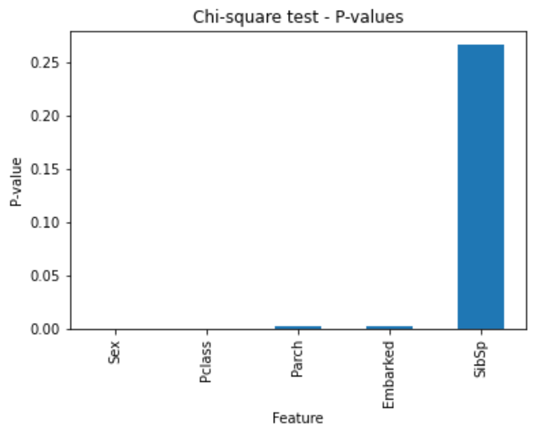
In [57]:
```
# HERE apply for desired  total  data

score = []

for feature in titanic.columns[:-1]:

    # create contingency table
    ct = pd.crosstab(titanic['Survived'], titanic[feature])

    # chi_test
    p_value = chi2_contingency(ct)[1]
    score.append(p_value)
```

In [58]:
```
pd.Series(score, index=titanic.columns[:-1]).sort_values(ascending=True).plot(kind='bar')


# Where ever 'P value' is small , they are important feature

# Here SEX , Pclass , Sibsp are Important Features
```

Out[58]:
```
<AxesSubplot:>
```



In [59]:
```
# Code for Sk.Learn

from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import chi2
import matplotlib.pyplot as plt

# assuming titanic is your DataFrame and 'Survived' is the target column

# Encode categorical variables
le = LabelEncoder()
titanic_encoded = titanic.apply(le.fit_transform)

X = titanic_encoded.drop('Survived', axis=1)
y = titanic_encoded['Survived']

# Calculate chi-squared stats
chi_scores = chi2(X, y)

# chi_scores[1] are the p-values of each feature.
p_values = pd.Series(chi_scores[1], index = X.columns)
p_values.sort_values(inplace = True)

# Plotting the p-values
p_values.plot.bar()

plt.title('Chi-square test - P-values')
plt.xlabel('Feature')
plt.ylabel('P-value')

plt.show()
```



### Disadvantages of Chi-Square

1. **Categorical Data Only**: The chi-square test can only be used with categorical variables. It is not suitable for continuous variables unless they have been discretized into categories, which can lead to loss of information.

2. **Independence of Observations**: The chi-square test assumes that the observations are independent of each other. This might not be the case in datasets where observations are related (e.g., time series data, nested data).

3. **Sufficient Sample Size**: Chi-square test requires a sufficiently large sample size. The results may not be reliable if the sample size is too small or if the frequency count in any category is too low (typically less than 5).

4. **No Variable Interactions**: Chi-square test, like other univariate feature selection methods, does not consider interactions between features. It might miss out on identifying important features that are significant in combination with other features.

# Advantages and Disadvantages of Filter Methods

## Advantages

1. **Simplicity**: Filter methods are generally straightforward and easy to understand. They involve calculating a statistic that measures the relevance of each feature, and selecting the top features based on this statistic.

2. **Speed**: These methods are usually computationally efficient. Because they evaluate each feature independently, they can be much faster than wrapper methods or embedded methods, which need to train a model to evaluate feature importance.

3. **Scalability**: Filter methods can handle a large number of features effectively because they don't involve any learning methods. This makes them suitable for high-dimensional datasets.

4. **Pre-processing Step**: They can serve as a pre-processing step for other feature selection methods. For instance, you could use a filter method to remove irrelevant features before applying a more computationally expensive method, such as a wrapper method.

## Disadvantages

1. **Lack of Feature Interaction**: Filter methods treat each feature individually and hence do not consider the interactions between features. They might miss out on identifying important features that don't appear significant individually but are significant in combination with other features.

2. **Model Agnostic**: Filter methods are agnostic to the machine learning model that will be used for the prediction. This means that the selected features might not necessarily contribute to the accuracy of the specific model you want to use.

3. **Statistical Measures Limitation**: The statistical measures used in these methods have their own limitations. For example, correlation is a measure of linear relationship and might not capture non-linear relationships effectively. Similarly, variance-based methods might keep features with high variance but low predictive power.

4. **Threshold Determination**: For some methods, determining the threshold to select features can be a bit subjective. For example, what constitutes "low" variance or "high" correlation might differ depending on the context or the specific dataset.

```
In [ ]:
```