In [1]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
import time
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
import pickle
import matplotlib.pyplot as plt
```

In [2]:
```python
def selectkbest(indep_X,dep_Y,n):
        test = SelectKBest(score_func=chi2, k=n)
        fit1= test.fit(indep_X,dep_Y)
        selectk_features = fit1.transform(indep_X)
        return selectk_features

def split_scalar(indep_X,dep_Y):
        X_train, X_test, y_train, y_test = train_test_split(indep_X, dep_Y, test_size = 0.25, random_state = 0)
        sc = StandardScaler()
        X_train = sc.fit_transform(X_train)
        X_test = sc.transform(X_test)
        return X_train, X_test, y_train, y_test

def cm_prediction(classifier,X_test):
     y_pred = classifier.predict(X_test)

        # Making the Confusion Matrix
     from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test, y_pred)

     from sklearn.metrics import accuracy_score
     from sklearn.metrics import classification_report
        #from sklearn.metrics import confusion_matrix
        #cm = confusion_matrix(y_test, y_pred)

     Accuracy=accuracy_score(y_test, y_pred )

     report=classification_report(y_test, y_pred)
     return  classifier,Accuracy,report,X_test,y_test,cm
```

In [3]:
```python
def logistic(X_train,y_train,X_test):
        # Fitting K-NN to the Training set
        from sklearn.linear_model import LogisticRegression
        classifier = LogisticRegression(random_state = 0)
        classifier.fit(X_train, y_train)
        classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm

def svm_linear(X_train,y_train,X_test):

        from sklearn.svm import SVC
        classifier = SVC(kernel = 'linear', random_state = 0)
        classifier.fit(X_train, y_train)
        classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm

def svm_NL(X_train,y_train,X_test):

        from sklearn.svm import SVC
        classifier = SVC(kernel = 'rbf', random_state = 0)
        classifier.fit(X_train, y_train)
        classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm

def Navie(X_train,y_train,X_test):
        # Fitting K-NN to the Training set
        from sklearn.naive_bayes import GaussianNB
        classifier = GaussianNB()
        classifier.fit(X_train, y_train)
        classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm


def knn(X_train,y_train,X_test):

        # Fitting K-NN to the Training set
        from sklearn.neighbors import KNeighborsClassifier
        classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
        classifier.fit(X_train, y_train)
        classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm

def Decision(X_train,y_train,X_test):

        # Fitting K-NN to the Training set
        from sklearn.tree import DecisionTreeClassifier
        classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
        classifier.fit(X_train, y_train)
        classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm


def random(X_train,y_train,X_test):

        # Fitting K-NN to the Training set
        from sklearn.ensemble import RandomForestClassifier
        classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
        classifier.fit(X_train, y_train)
        classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm
def selectk_Classification(acclog,accsvml,accsvmnl,accknn,accnav,accdes,accrf):

    dataframe=pd.DataFrame(index=['ChiSquare'],columns=['Logistic','SVMl','SVMnl','KNN','Navie','Decision','Random'])
    for number,idex in enumerate(dataframe.index):
        dataframe['Logistic'][idex]=acclog[number]
        dataframe['SVMl'][idex]=accsvml[number]
        dataframe['SVMnl'][idex]=accsvmnl[number]
        dataframe['KNN'][idex]=accknn[number]
        dataframe['Navie'][idex]=accnav[number]
        dataframe['Decision'][idex]=accdes[number]
        dataframe['Random'][idex]=accrf[number]
    return dataframe
```

In [4]:
```python
dataset=pd.read_csv("ad_click_dataset.csv")
dataset
```

Out[4]:

	id	full_name	age	gender	device_type	ad_position	browsing_history	time_of_day	click
0	670	User670	22.0	NaN	Desktop	Top	Shopping	Afternoon	1
1	3044	User3044	NaN	Male	Desktop	Top	NaN	NaN	1
2	5912	User5912	41.0	Non-Binary	NaN	Side	Education	Night	1
3	5418	User5418	34.0	Male	NaN	NaN	Entertainment	Evening	1
4	9452	User9452	39.0	Non-Binary	NaN	NaN	Social Media	Morning	0
...	...	...	...	...	...	...	...	...	...
9995	8510	User8510	NaN	NaN	Mobile	Top	Education	NaN	0
9996	7843	User7843	NaN	Female	Desktop	Bottom	Entertainment	NaN	0
9997	3914	User3914	NaN	Male	Mobile	Side	NaN	Morning	0
9998	7924	User7924	NaN	NaN	Desktop	NaN	Shopping	Morning	1
9999	3056	User3056	44.0	Male	Tablet	Top	Social Media	Morning	0

10000 rows × 9 columns

In [5]:
```python
dataset=pd.get_dummies(dataset,drop_first=True)
dataset.dropna(inplace=True)
dataset
```

Out[5]:

	id	age	click	full_name_User100	full_name_User1000	full_name_User10000	full_name_User1001	full_name_User1002	full_name_User1011	full_name_User1012	...	device_type_Tablet	ad_position_Side	ad_position_Top	browsing_history_Entertainment	browsing_history_News	bro
0	670	22.0	1	False	False	False	False	False	False	False	...	False	False	True	False	False	
2	5912	41.0	1	False	False	False	False	False	False	False	...	False	True	False	False	False	
3	5418	34.0	1	False	False	False	False	False	False	False	...	False	False	False	True	False	
4	9452	39.0	0	False	False	False	False	False	False	False	...	False	False	False	False	False	
6	7808	26.0	1	False	False	False	False	False	False	False	...	False	False	True	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
9987	2876	23.0	1	False	False	False	False	False	False	False	...	True	False	False	False	False	
9988	2713	52.0	0	False	False	False	False	False	False	False	...	False	True	False	False	False	
9990	9540	64.0	0	False	False	False	False	False	False	False	...	False	True	False	False	False	
9993	503	43.0	1	False	False	False	False	False	False	False	...	False	False	False	False	False	
9999	3056	44.0	0	False	False	False	False	False	False	False	...	True	False	True	False	False	

5234 rows × 4015 columns

In [6]:
```python
dataset.columns
```

Out[6]:
```
Index(['id', 'age', 'click', 'full_name_User100', 'full_name_User1000',
       'full_name_User10000', 'full_name_User1001', 'full_name_User1002',
       'full_name_User1011', 'full_name_User1012',
       ...
       'device_type_Tablet', 'ad_position_Side', 'ad_position_Top',
       'browsing_history_Entertainment', 'browsing_history_News',
       'browsing_history_Shopping', 'browsing_history_Social Media',
       'time_of_day_Evening', 'time_of_day_Morning', 'time_of_day_Night'],
      dtype='object', length=4015)
```

In [7]:
```python
dataset["click"].value_counts()
```

Out[7]:
```
click
1    3351
0    1883
Name: count, dtype: int64
```

In [9]:
```python
indep_X=dataset[['id','age','full_name_User100','full_name_User1000','full_name_User10000','full_name_User1001','full_name_User1002','full_name_User1011','full_name_User1012','device_type_Tablet','ad_position_Side','ad_position_Top','
dep_Y=dataset['click']
```

In [17]:
```python
kbest=selectkbest(indep_X,dep_Y,5)
acclog=[]
accsvml=[]
accsvmnl=[]
accknn=[]
accnav=[]
accdes=[]
accrf=[]
kbest
```

Out[17]:
```
array([[6.700e+02, 2.200e+01, 0.000e+00, 0.000e+00, 0.000e+00],
       [5.912e+03, 4.100e+01, 0.000e+00, 0.000e+00, 0.000e+00],
       [5.418e+03, 3.400e+01, 1.000e+00, 1.000e+00, 0.000e+00],
       ...,
       [9.540e+03, 6.400e+01, 0.000e+00, 0.000e+00, 0.000e+00],
       [5.030e+02, 4.300e+01, 0.000e+00, 0.000e+00, 1.000e+00],
       [3.056e+03, 4.400e+01, 0.000e+00, 0.000e+00, 1.000e+00]])
```

In [18]:
```python
X_train, X_test, y_train, y_test=split_scalar(kbest,dep_Y)

classifier,Accuracy,report,X_test,y_test,cm=logistic(X_train,y_train,X_test)
acclog.append(Accuracy)

classifier,Accuracy,report,X_test,y_test,cm=svm_linear(X_train,y_train,X_test)
accsvml.append(Accuracy)

classifier,Accuracy,report,X_test,y_test,cm=svm_NL(X_train,y_train,X_test)
accsvmnl.append(Accuracy)

classifier,Accuracy,report,X_test,y_test,cm=knn(X_train,y_train,X_test)
accknn.append(Accuracy)

classifier,Accuracy,report,X_test,y_test,cm=Navie(X_train,y_train,X_test)
accnav.append(Accuracy)

classifier,Accuracy,report,X_test,y_test,cm=Decision(X_train,y_train,X_test)
accdes.append(Accuracy)

classifier,Accuracy,report,X_test,y_test,cm=random(X_train,y_train,X_test)
accrf.append(Accuracy)
```

```
C:\Users\SowmiGanesh\anaconda3\envs\aiml\Lib\site-packages\sklearn\metrics\_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` paramete
r to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\SowmiGanesh\anaconda3\envs\aiml\Lib\site-packages\sklearn\metrics\_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` paramete
r to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\SowmiGanesh\anaconda3\envs\aiml\Lib\site-packages\sklearn\metrics\_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` paramete
r to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\SowmiGanesh\anaconda3\envs\aiml\Lib\site-packages\sklearn\metrics\_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` paramete
r to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\SowmiGanesh\anaconda3\envs\aiml\Lib\site-packages\sklearn\metrics\_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` paramete
r to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\SowmiGanesh\anaconda3\envs\aiml\Lib\site-packages\sklearn\metrics\_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` paramete
r to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

In [13]:
```python
result=selectk_Classification(acclog,accsvml,accsvmnl,accknn,accnav,accdes,accrf)
result
#4
```

```
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:65: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['Logistic'][idex]=acclog[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:66: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['SVMl'][idex]=accsvml[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:67: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['SVMnl'][idex]=accsvmnl[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:68: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['KNN'][idex]=accknn[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:69: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['Navie'][idex]=accnav[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:70: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['Decision'][idex]=accdes[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:71: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['Random'][idex]=accrf[number]
```

Out[13]:

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
ChiSquare	0.61039	0.61039	0.611917	0.781513	0.613445	0.921314	0.919786

In [16]:
```python
result=selectk_Classification(acclog,accsvml,accsvmnl,accknn,accnav,accdes,accrf
result
#3
```

```
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:65: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['Logistic'][idex]=acclog[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:66: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['SVMl'][idex]=accsvml[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:67: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['SVMnl'][idex]=accsvmnl[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:68: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['KNN'][idex]=accknn[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:69: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['Navie'][idex]=accnav[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:70: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['Decision'][idex]=accdes[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:71: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['Random'][idex]=accrf[number]
```

Out[16]:

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
ChiSquare	0.61039	0.61039	0.611154	0.808251	0.611917	0.927426	0.92437

In [19]:
```python
result=selectk_Classification(acclog,accsvml,accsvmnl,accknn,accnav,accdes,accrf)
result
#5
```

```
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:65: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['Logistic'][idex]=acclog[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:66: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['SVMl'][idex]=accsvml[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:67: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['SVMnl'][idex]=accsvmnl[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:68: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['KNN'][idex]=accknn[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:69: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['Navie'][idex]=accnav[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:70: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['Decision'][idex]=accdes[number]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_9300\2796833036.py:71: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Ser
ies, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dataframe['Random'][idex]=accrf[number]
```

Out[19]:

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
ChiSquare	0.61039	0.61039	0.614209	0.754775	0.613445	0.912911	0.896104

In [ ]:

In [ ]: