```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.api.types import is_numeric_dtype
import warnings
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.tree  import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier, VotingClassifier, GradientBoostingClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.naive_bayes import BernoulliNB
from lightgbm import LGBMClassifier
from sklearn.feature_selection import RFE
import itertools
from xgboost import XGBClassifier
from tabulate import tabulate

train = pd.read_csv('Train_data.csv')

test = pd.read_csv('Test_data.csv')

train.head()
```

```
   duration protocol_type   service flag  src_bytes  dst_bytes
land  \
0         0           tcp  ftp_data   SF        491          0     0

1         0           udp     other   SF        146          0     0

2         0           tcp   private   S0          0          0     0

3         0           tcp      http   SF        232       8153     0

4         0           tcp      http   SF        199        420     0


   wrong_fragment  urgent  hot  ...  dst_host_srv_count  \
0               0       0    0  ...                  25
1               0       0    0  ...                   1
2               0       0    0  ...                  26
3               0       0    0  ...                 255
4               0       0    0  ...                 255

   dst_host_same_srv_rate  dst_host_diff_srv_rate  \
0                    0.17                    0.03
1                    0.00                    0.60
```

```
2                    0.10                    0.05
3                    1.00                    0.00
4                    1.00                    0.00

   dst_host_same_src_port_rate  dst_host_srv_diff_host_rate  \
0                         0.17                         0.00
1                         0.88                         0.00
2                         0.00                         0.00
3                         0.03                         0.04
4                         0.00                         0.00

   dst_host_serror_rate  dst_host_srv_serror_rate
dst_host_rerror_rate  \
0                  0.00                      0.00
0.05
1                  0.00                      0.00
0.00
2                  1.00                      1.00
0.00
3                  0.03                      0.01
0.00
4                  0.00                      0.00
0.00

   dst_host_srv_rerror_rate    class
0                      0.00   normal
1                      0.00   normal
2                      0.00  anomaly
3                      0.01   normal
4                      0.00   normal

[5 rows x 42 columns]

train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25192 entries, 0 to 25191
Data columns (total 42 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   duration                     25192 non-null  int64
 1   protocol_type                25192 non-null  object
 2   service                      25192 non-null  object
 3   flag                         25192 non-null  object
 4   src_bytes                    25192 non-null  int64
 5   dst_bytes                    25192 non-null  int64
 6   land                         25192 non-null  int64
 7   wrong_fragment               25192 non-null  int64
 8   urgent                       25192 non-null  int64
 9   hot                          25192 non-null  int64
```

```
 10  num_failed_logins            25192 non-null  int64
 11  logged_in                    25192 non-null  int64
 12  num_compromised              25192 non-null  int64
 13  root_shell                   25192 non-null  int64
 14  su_attempted                 25192 non-null  int64
 15  num_root                     25192 non-null  int64
 16  num_file_creations           25192 non-null  int64
 17  num_shells                   25192 non-null  int64
 18  num_access_files             25192 non-null  int64
 19  num_outbound_cmds            25192 non-null  int64
 20  is_host_login                25192 non-null  int64
 21  is_guest_login               25192 non-null  int64
 22  count                        25192 non-null  int64
 23  srv_count                    25192 non-null  int64
 24  serror_rate                  25192 non-null  float64
 25  srv_serror_rate              25192 non-null  float64
 26  rerror_rate                  25192 non-null  float64
 27  srv_rerror_rate              25192 non-null  float64
 28  same_srv_rate                25192 non-null  float64
 29  diff_srv_rate                25192 non-null  float64
 30  srv_diff_host_rate           25192 non-null  float64
 31  dst_host_count               25192 non-null  int64
 32  dst_host_srv_count           25192 non-null  int64
 33  dst_host_same_srv_rate       25192 non-null  float64
 34  dst_host_diff_srv_rate       25192 non-null  float64
 35  dst_host_same_src_port_rate  25192 non-null  float64
 36  dst_host_srv_diff_host_rate  25192 non-null  float64
 37  dst_host_serror_rate         25192 non-null  float64
 38  dst_host_srv_serror_rate     25192 non-null  float64
 39  dst_host_rerror_rate         25192 non-null  float64
 40  dst_host_srv_rerror_rate     25192 non-null  float64
 41  class                        25192 non-null  object
dtypes: float64(15), int64(23), object(4)
memory usage: 8.1+ MB

train.describe()

           duration     src_bytes     dst_bytes           land
wrong_fragment  \
count  25192.000000  2.519200e+04  2.519200e+04  25192.000000
25192.000000
mean     305.054104  2.433063e+04  3.491847e+03      0.000079
0.023738
std     2686.555640  2.410805e+06  8.883072e+04      0.008910
0.260221
min        0.000000  0.000000e+00  0.000000e+00      0.000000
0.000000
25%        0.000000  0.000000e+00  0.000000e+00      0.000000
0.000000
50%        0.000000  4.400000e+01  0.000000e+00      0.000000
```

```
0.000000
75%        0.000000  2.790000e+02  5.302500e+02      0.000000
0.000000
max    42862.000000  3.817091e+08  5.151385e+06      1.000000
3.000000

             urgent           hot  num_failed_logins     logged_in  \
count  25192.00000  25192.000000       25192.000000  25192.000000
mean       0.00004      0.198039           0.001191      0.394768
std        0.00630      2.154202           0.045418      0.488811
min        0.00000      0.000000           0.000000      0.000000
25%        0.00000      0.000000           0.000000      0.000000
50%        0.00000      0.000000           0.000000      0.000000
75%        0.00000      0.000000           0.000000      1.000000
max        1.00000     77.000000           4.000000      1.000000

       num_compromised  ...  dst_host_count  dst_host_srv_count  \
count     25192.000000  ...    25192.000000        25192.000000
mean          0.227850  ...      182.532074          115.063036
std          10.417352  ...       98.993895          110.646850
min           0.000000  ...        0.000000            0.000000
25%           0.000000  ...       84.000000           10.000000
50%           0.000000  ...      255.000000           61.000000
75%           0.000000  ...      255.000000          255.000000
max         884.000000  ...      255.000000          255.000000

       dst_host_same_srv_rate  dst_host_diff_srv_rate  \
count            25192.000000            25192.000000
mean                 0.519791                0.082539
std                  0.448944                0.187191
min                  0.000000                0.000000
25%                  0.050000                0.000000
50%                  0.510000                0.030000
75%                  1.000000                0.070000
max                  1.000000                1.000000

       dst_host_same_src_port_rate  dst_host_srv_diff_host_rate  \
count                  25192.000000                 25192.000000
mean                       0.147453                     0.031844
std                        0.308367                     0.110575
min                        0.000000                     0.000000
25%                        0.000000                     0.000000
50%                        0.000000                     0.000000
75%                        0.060000                     0.020000
max                        1.000000                     1.000000

       dst_host_serror_rate  dst_host_srv_serror_rate
dst_host_rerror_rate  \
count          25192.000000              25192.000000
25192.000000
```

```
mean                 0.285800                      0.279846
0.117800
std                  0.445316                      0.446075
0.305869
min                  0.000000                      0.000000
0.000000
25%                  0.000000                      0.000000
0.000000
50%                  0.000000                      0.000000
0.000000
75%                  1.000000                      1.000000
0.000000
max                  1.000000                      1.000000
1.000000

       dst_host_srv_rerror_rate
count              25192.000000
mean                   0.118769
std                    0.317333
min                    0.000000
25%                    0.000000
50%                    0.000000
75%                    0.000000
max                    1.000000

[8 rows x 38 columns]
```

```
train.describe(include="object")
```

```
       protocol_type service   flag   class
count          25192   25192  25192   25192
unique             3      66     11       2
top              tcp    http     SF  normal
freq           20526    8003  14973   13449
```

```
train.shape
```

```
(25192, 42)
```

```
train.isnull().sum()
```

```
duration                        0
protocol_type                   0
service                         0
flag                            0
src_bytes                       0
dst_bytes                       0
land                            0
wrong_fragment                  0
urgent                          0
hot                             0
```

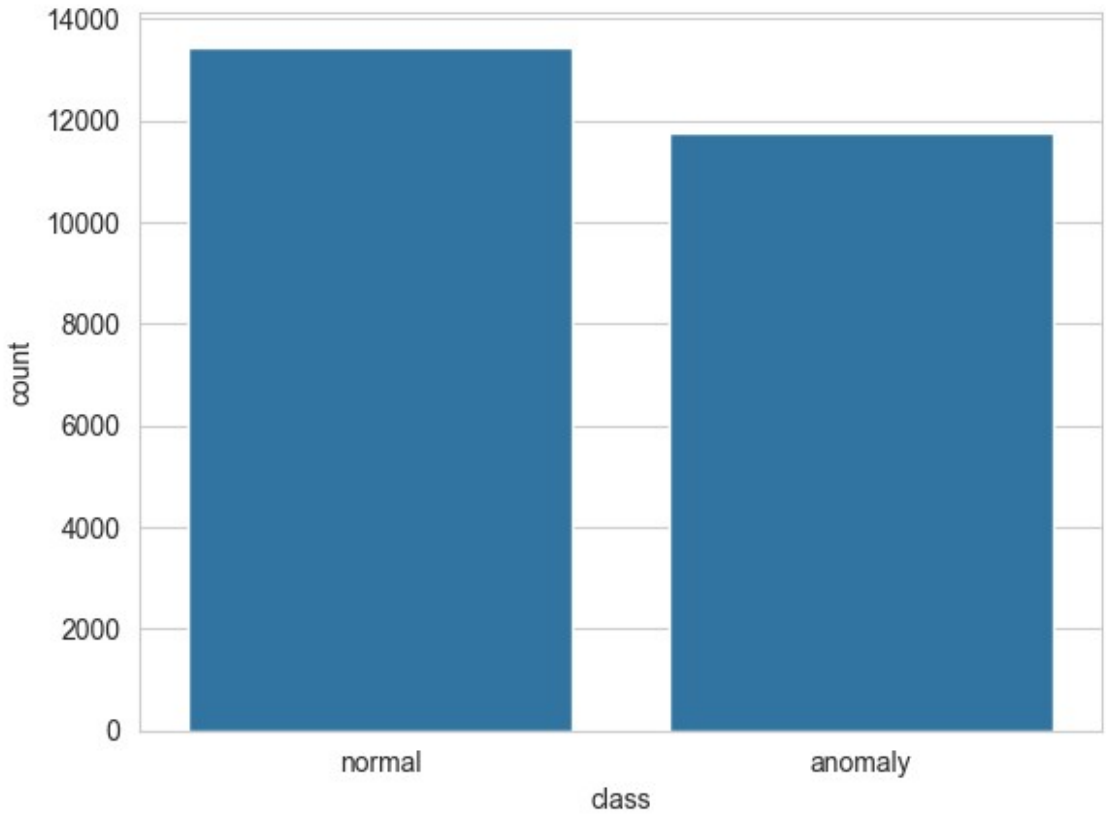
```
num_failed_logins               0
logged_in                       0
num_compromised                 0
root_shell                      0
su_attempted                    0
num_root                        0
num_file_creations              0
num_shells                      0
num_access_files                0
num_outbound_cmds               0
is_host_login                   0
is_guest_login                  0
count                           0
srv_count                       0
serror_rate                     0
srv_serror_rate                 0
rerror_rate                     0
srv_rerror_rate                 0
same_srv_rate                   0
diff_srv_rate                   0
srv_diff_host_rate              0
dst_host_count                  0
dst_host_srv_count              0
dst_host_same_srv_rate          0
dst_host_diff_srv_rate          0
dst_host_same_src_port_rate     0
dst_host_srv_diff_host_rate     0
dst_host_serror_rate            0
dst_host_srv_serror_rate        0
dst_host_rerror_rate            0
dst_host_srv_rerror_rate        0
class                           0
dtype: int64
```

```
total = train.shape[0]
missing_columns = [col for col in train.columns if
train[col].isnull().sum() > 0]
for col in missing_columns:
    null_count = train[col].isnull().sum()
    per = (null_count/total) * 100
    print(f"{col}: {null_count} ({round(per, 3)}%)")

print(f"Number of duplicate rows: {train.duplicated().sum()}")
```

```
Number of duplicate rows: 0
```

```
sns.countplot(x=train['class'])
```

```
<Axes: xlabel='class', ylabel='count'>
```




```
print("Class Training Classification")
print(train['class'].value_counts())
```

```
Class Training Classification
class
1    13449
0    11743
Name: count, dtype: int64
```

```
#The primary purpose of this function is to preprocess the data by
converting categorical variables into numerical form,
#which is a necessary step for many machine learning algorithms that
require numeric input.
def le(df):
    for col in df.columns:
        if df[col].dtype == 'object':
                label_encoder = LabelEncoder()
                df[col] = label_encoder.fit_transform(df[col])

le(train)
le(test)

train.drop(['num_outbound_cmds'], axis=1, inplace=True)
test.drop(['num_outbound_cmds'], axis=1, inplace=True)
#dropping unnecessary columns similar like in database drop column
```

```
train.head()
```

```
   duration  protocol_type  service  flag  src_bytes  dst_bytes  land
\
0         0              1       19     9        491          0     0

1         0              2       41     9        146          0     0

2         0              1       46     5          0          0     0

3         0              1       22     9        232       8153     0

4         0              1       22     9        199        420     0


   wrong_fragment  urgent  hot  ...  dst_host_srv_count  \
0               0       0    0  ...                  25
1               0       0    0  ...                   1
2               0       0    0  ...                  26
3               0       0    0  ...                 255
4               0       0    0  ...                 255

   dst_host_same_srv_rate  dst_host_diff_srv_rate  \
0                    0.17                    0.03
1                    0.00                    0.60
2                    0.10                    0.05
3                    1.00                    0.00
4                    1.00                    0.00

   dst_host_same_src_port_rate  dst_host_srv_diff_host_rate  \
0                         0.17                         0.00
1                         0.88                         0.00
2                         0.00                         0.00
3                         0.03                         0.04
4                         0.00                         0.00

   dst_host_serror_rate  dst_host_srv_serror_rate
dst_host_rerror_rate  \
0                  0.00                     0.00
0.05
1                  0.00                     0.00
0.00
2                  1.00                     1.00
0.00
3                  0.03                     0.01
0.00
4                  0.00                     0.00
0.00

   dst_host_srv_rerror_rate  class
```

```
0                       0.00      1
1                       0.00      1
2                       0.00      0
3                       0.01      1
4                       0.00      1

[5 rows x 41 columns]
```

```
X_train = train.drop(['class'], axis=1)
Y_train = train['class']

rfc = RandomForestClassifier()

rfe = RFE(rfc, n_features_to_select=10)
rfe = rfe.fit(X_train, Y_train)

feature_map = [(i, v) for i, v in
itertools.zip_longest(rfe.get_support(), X_train.columns)]
selected_features = [v for i, v in feature_map if i==True]

selected_features
#A feature selection method that fits a model and removes the weakest
feature(s) until the specified number of features is reached.
#It recursively removes features, builds the model on the remaining
features, and repeats the process until the desired number of features
is selected.
```

```
['protocol_type',
 'flag',
 'src_bytes',
 'dst_bytes',
 'count',
 'same_srv_rate',
 'diff_srv_rate',
 'dst_host_srv_count',
 'dst_host_same_srv_rate',
 'dst_host_same_src_port_rate']
```

```
X_train = X_train[selected_features]

scale = StandardScaler()
X_train = scale.fit_transform(X_train)
test = scale.fit_transform(test)

x_train, x_test, y_train, y_test = train_test_split(X_train, Y_train,
train_size=0.70, random_state=2)

x_train.shape
```

```
(17634, 10)
```

```
x_test.shape
```

```
(7558, 10)
```

```
y_train.shape
```

```
(17634,)
```

```
y_test.shape
```

```
(7558,)
```

```
import time
#Logistic regression is a statistical method used for binary
classification problems, where the goal is to predict one of two
possible outcomes.
#It models the probability that a given input belongs to a certain
class.

from sklearn.linear_model import LogisticRegression

clfl = LogisticRegression(max_iter = 1200000)
start_time = time.time()
clfl.fit(x_train, y_train.values.ravel())
end_time = time.time()
print("Training time: ", end_time-start_time)


Training time:  0.048136234283447266

start_time = time.time()
y_test_pred = clfl.predict(x_train)
end_time = time.time()
print("Testing time: ", end_time-start_time)

Testing time:  0.0044634342193603516

lg_model = LogisticRegression(random_state = 42)
lg_model.fit(x_train, y_train)

LogisticRegression(random_state=42)

lg_train, lg_test = lg_model.score(x_train , y_train),
lg_model.score(x_test , y_test)

print(f"Training Score: {lg_train}")
print(f"Test Score: {lg_test}")

Training Score: 0.9417035272768516
Test Score: 0.938872717650172

import optuna
optuna.logging.set_verbosity(optuna.logging.WARNING)
```

```
def objective(trial):
    n_neighbors = trial.suggest_int('KNN_n_neighbors', 2, 16,
log=False)
    classifier_obj = KNeighborsClassifier(n_neighbors=n_neighbors)
    classifier_obj.fit(x_train, y_train)
    accuracy = classifier_obj.score(x_test, y_test)
    return accuracy
#KNN Accuracy Test

study_KNN = optuna.create_study(direction='maximize')
study_KNN.optimize(objective, n_trials=1)
print(study_KNN.best_trial)
```

```
FrozenTrial(number=0, state=1, values=[0.9801534797565493],
datetime_start=datetime.datetime(2024, 6, 9, 14, 28, 11, 755981),
datetime_complete=datetime.datetime(2024, 6, 9, 14, 28, 13, 473506),
params={'KNN_n_neighbors': 9}, user_attrs={}, system_attrs={},
intermediate_values={}, distributions={'KNN_n_neighbors':
IntDistribution(high=16, log=False, low=2, step=1)}, trial_id=0,
value=None)
```

```
KNN_model =
KNeighborsClassifier(n_neighbors=study_KNN.best_trial.params['KNN_n_ne
ighbors'])
KNN_model.fit(x_train, y_train)

KNN_train, KNN_test = KNN_model.score(x_train, y_train),
KNN_model.score(x_test, y_test)

print(f"Train Score: {KNN_train}")
print(f"Test Score: {KNN_test}")
#KNN Test and Train Score
```

```
Train Score: 0.9835544969944425
Test Score: 0.9801534797565493
```

```
from sklearn.tree import DecisionTreeClassifier

clfd = DecisionTreeClassifier(criterion ="entropy", max_depth = 4)
start_time = time.time()
clfd.fit(x_train, y_train.values.ravel())
end_time = time.time()
print("Training time: ", end_time-start_time)
```

```
Training time:  0.045543670654296875
```

```
start_time = time.time()
y_test_pred = clfd.predict(x_train)
end_time = time.time()
print("Testing time: ", end_time-start_time)
```

```
Testing time:  0.042575836181640625
```

```
def objective(trial):
    dt_max_depth = trial.suggest_int('dt_max_depth', 2, 32, log=False)
    dt_max_features = trial.suggest_int('dt_max_features', 2, 10,
log=False)
    classifier_obj = DecisionTreeClassifier(max_features =
dt_max_features, max_depth = dt_max_depth)
    classifier_obj.fit(x_train, y_train)
    accuracy = classifier_obj.score(x_test, y_test)
    return accuracy

study_dt = optuna.create_study(direction='maximize')
study_dt.optimize(objective, n_trials=30)
print(study_dt.best_trial)
```

```
FrozenTrial(number=26, state=1, values=[0.9951045250066155],
datetime_start=datetime.datetime(2024, 6, 9, 14, 29, 20, 660724),
datetime_complete=datetime.datetime(2024, 6, 9, 14, 29, 20, 742957),
params={'dt_max_depth': 15, 'dt_max_features': 9}, user_attrs={},
system_attrs={}, intermediate_values={},
distributions={'dt_max_depth': IntDistribution(high=32, log=False,
low=2, step=1), 'dt_max_features': IntDistribution(high=10, log=False,
low=2, step=1)}, trial_id=26, value=None)
```

```
dt = DecisionTreeClassifier(max_features =
study_dt.best_trial.params['dt_max_features'], max_depth =
study_dt.best_trial.params['dt_max_depth'])
dt.fit(x_train, y_train)

dt_train, dt_test = dt.score(x_train, y_train), dt.score(x_test,
y_test)

print(f"Train Score: {dt_train}")
print(f"Test Score: {dt_test}")
```

```
Train Score: 1.0
Test Score: 0.9941783540619211
```

```
data = [["KNN", KNN_train, KNN_test],
        ["Logistic Regression", lg_train, lg_test],
        ["Decision Tree", dt_train, dt_test]]

col_names = ["Model", "Train Score", "Test Score"]
print(tabulate(data, headers=col_names, tablefmt="fancy_grid"))
```

```
╒═══════════════════╤═══════════════╤══════════════╕
│ Model             │   Train Score │   Test Score │
╞═══════════════════╪═══════════════╪══════════════╡
│ KNN               │      0.983554 │     0.980153 │
```

Logistic Regression	0.941704	0.938873
Decision Tree	1	0.994178

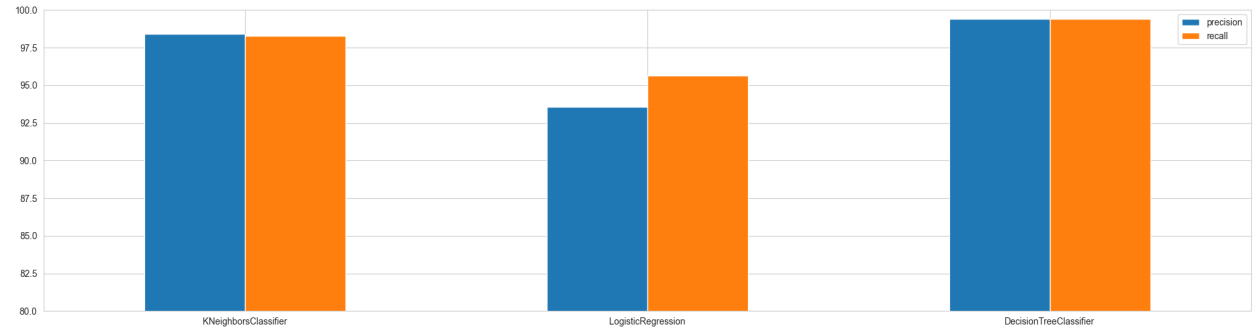
```
SEED = 42

# Decision Tree Model
dtc = DecisionTreeClassifier()

# KNN
knn = KNeighborsClassifier()

# LOGISTIC REGRESSION MODEL

lr = LogisticRegression()

from sklearn.model_selection import cross_val_score
models = {}
models['KNeighborsClassifier']= knn
models['LogisticRegression']= lr
models['DecisionTreeClassifier']= dtc

scores = {}
for name in models:
  scores[name]={}
  for scorer in ['precision','recall']:
    scores[name][scorer] = cross_val_score(models[name], x_train,
y_train, cv=10, scoring=scorer)

def line(name):
  return '*'*(25-len(name)//2)

for name in models:
  print(line(name), name, 'Model Validation', line(name))

  for scorer in ['precision','recall']:
    mean = round(np.mean(scores[name][scorer])*100,2)
    stdev = round(np.std(scores[name][scorer])*100,2)
    print ("Mean {}:".format(scorer),"\n", mean,"%", "+-",stdev)
    print()
```

```
*************** KNeighborsClassifier Model Validation ***************
Mean precision:
 98.39 % +- 0.41

Mean recall:
 98.3 % +- 0.44

*************** LogisticRegression Model Validation ***************
Mean precision:
```

```
 93.57 % +- 0.63

Mean recall:
 95.65 % +- 0.59

************** DecisionTreeClassifier Model Validation **************
Mean precision:
 99.42 % +- 0.3

Mean recall:
 99.42 % +- 0.23
```

```
for name in models:
    for scorer in ['precision','recall']:
        scores[name][scorer] = scores[name][scorer].mean()
scores = pd.DataFrame(scores).transpose() * 100
scores.plot(kind = "bar",  ylim=[80,100], figsize=(24,6), rot=0)
```

```
<Axes: >
```




```
models = {}
models['KNeighborsClassifier']= knn
models['LogisticRegression']= lr
models['DecisionTreeClassifier']= dtc

preds={}
for name in models:
    models[name].fit(x_train, y_train)
    preds[name] = models[name].predict(x_test)
print("Predictions complete.")
```

```
Predictions complete.
```

```
from sklearn.metrics import confusion_matrix, classification_report,
f1_score
def line(name,sym="*"):
    return sym*(25-len(name)//2)
target_names=["normal","anamoly"]
for name in models:
```

```
    print(line(name), name, 'Model Testing', line(name))
    print(confusion_matrix(y_test, preds[name]))
    print(line(name,'-'))
    print(classification_report(y_test, preds[name],
target_names=target_names))
```

```
*************** KNeighborsClassifier Model Testing ***************
[[3423   75]
 [  57 4003]]
---------------
              precision    recall  f1-score   support

      normal       0.98      0.98      0.98      3498
     anamoly       0.98      0.99      0.98      4060

    accuracy                           0.98      7558
   macro avg       0.98      0.98      0.98      7558
weighted avg       0.98      0.98      0.98      7558

*************** LogisticRegression Model Testing ***************
[[3223  275]
 [ 187 3873]]
---------------
              precision    recall  f1-score   support

      normal       0.95      0.92      0.93      3498
     anamoly       0.93      0.95      0.94      4060

    accuracy                           0.94      7558
   macro avg       0.94      0.94      0.94      7558
weighted avg       0.94      0.94      0.94      7558

************** DecisionTreeClassifier Model Testing **************
[[3475   23]
 [  21 4039]]
--------------
              precision    recall  f1-score   support

      normal       0.99      0.99      0.99      3498
     anamoly       0.99      0.99      0.99      4060

    accuracy                           0.99      7558
   macro avg       0.99      0.99      0.99      7558
weighted avg       0.99      0.99      0.99      7558
```

```
f1s = {}
for name in models:
    f1s[name]=f1_score(y_test, preds[name])
f1s=pd.DataFrame(f1s.values(),index=f1s.keys(),columns=["F1-
```

```
score"])*100
f1s.plot(kind = "bar",  ylim=[80,100], figsize=(10,6), rot=0)
```

```
<Axes: >
```