```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
print("ANIRUDH SHUKLA 00619011921")
```

```
ANIRUDH SHUKLA 00619011921
```

```
# IMPORTING LIBRARIES
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# DATA LOADING
# LIST OF COLUMNS NAME
columnsName = ["id_1","id_2","cmp_fname_c1","cmp_fname_c2","cmp_lname_c1","cmp_lname_c2",
               "cmp_sex","cmp_bd","cmp_bm","cmp_by","cmp_plz","is_match"]
print(len(columnsName))
```

```
12
```

```
# INITIAL FILE PATH
file_path = "/content/drive/MyDrive/donation/"

# ALL SUBJECT DATA FILE NAME (IN LIST)
file_path_loc = ["block_1.csv", "block_2.csv", "block_3.csv", "block_4.csv", "block_5.csv",
                 "block_6.csv", "block_7.csv", "block_8.csv", "block_9.csv", "block_10.csv",]

# EMPTY LIST FOR STORING DATAFRAME
dataframes_list = []

for file in file_path_loc:
  # READING DATA FILE AND SET COLUMN NAME AS ABOVE LIST
    df = pd.read_csv(file_path + file, delimiter=',')
    dataframes_list.append(df)

# MAKING A SINGLE DATAFRAME FOR ALL ABOVE SUBJECT DATA
df = pd.concat(dataframes_list)

# Reset the index of the concatenated DataFrame
concatenated_df = df.reset_index(drop=True)

print(df)
```

```
          id_1   id_2       cmp_fname_c1 cmp_fname_c2  cmp_lname_c1  \
0       37291  53113  0.833333333333333            ?      1.000000
1       39086  47614                  1            ?      1.000000
2       70031  70237                  1            ?      1.000000
3       84795  97439                  1            ?      1.000000
4       36950  42116                  1            ?      1.000000
...       ...    ...                ...          ...           ...
574908  32517  73116                  1            ?      0.222222
574909  67707  83757  0.111111111111111            ?      1.000000
574910  53258  91808                  1            ?      0.000000
574911  31865  85285                  1            ?      0.111111
574912  33119  76399                  1            ?      0.000000

       cmp_lname_c2  cmp_sex cmp_bd cmp_bm cmp_by cmp_plz  is_match
0                 ?        1      1      1      1       0      True
1                 ?        1      1      1      1       1      True
2                 ?        1      1      1      1       1      True
3                 ?        1      1      1      1       1      True
4                 1        1      1      1      1       1      True
...             ...      ...    ...    ...    ...     ...       ...
574908            ?        1      0      1      0       0     False
574909            ?        1      0      0      0       0     False
574910            ?        1      0      0      1       0     False
574911            ?        1      0      1      0       0     False
574912            ?        1      0      1      0       0     False

[5749132 rows x 12 columns]
```

```
# REPLACING ? BY NAN VALUES
df.replace('?',np.nan,inplace=True)
```

```
df
```

	id_1	id_2	cmp_fname_c1	cmp_fname_c2	cmp_lname_c1	cmp_lname_c2	cmp_sex	cmp_bd	cmp_bm	cmp_by	cmp_plz
0	37291	53113	0.833333333333333	NaN	1.000000	NaN	1	1	1	1	0
1	39086	47614	1	NaN	1.000000	NaN	1	1	1	1	1
2	70031	70237	1	NaN	1.000000	NaN	1	1	1	1	1
3	84795	97439	1	NaN	1.000000	NaN	1	1	1	1	1
4	36950	42116	1	NaN	1.000000	1	1	1	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...
574908	32517	73116	1	NaN	0.222222	NaN	1	0	1	0	0
574909	67707	83757	0.111111111111111	NaN	1.000000	NaN	1	0	0	0	0
574910	53258	91808	1	NaN	0.000000	NaN	1	0	0	1	0
574911	31865	85285	1	NaN	0.111111	NaN	1	0	1	0	0
574912	33119	76399	1	NaN	0.000000	NaN	1	0	1	0	0

5749132 rows × 12 columns

```
df.shape
```

```
(5749132, 12)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5749132 entries, 0 to 574912
Data columns (total 12 columns):
 #   Column        Dtype  
---  ------        -----  
 0   id_1          int64  
 1   id_2          int64  
 2   cmp_fname_c1  object 
 3   cmp_fname_c2  object 
 4   cmp_lname_c1  float64
 5   cmp_lname_c2  object 
 6   cmp_sex       int64  
 7   cmp_bd        object 
 8   cmp_bm        object 
 9   cmp_by        object 
 10  cmp_plz       object 
 11  is_match      bool   
dtypes: bool(1), float64(1), int64(3), object(7)
memory usage: 531.8+ MB
```

```
# CONVERTING OBJECT DATA TYPES INTO INT OR FLOAT DATA TYPE
df["cmp_fname_c1"] = pd.to_numeric(df["cmp_fname_c1"])
df["cmp_fname_c2"] = pd.to_numeric(df["cmp_fname_c2"])
df["cmp_lname_c2"] = pd.to_numeric(df["cmp_fname_c2"])
df["cmp_bd"] = pd.to_numeric(df["cmp_bd"])
df["cmp_bm"] = pd.to_numeric(df["cmp_bm"])
df["cmp_by"] = pd.to_numeric(df["cmp_by"])
df["cmp_plz"] = pd.to_numeric(df["cmp_plz"])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5749132 entries, 0 to 574912
Data columns (total 12 columns):
 #   Column        Dtype  
---  ------        -----  
 0   id_1          int64  
 1   id_2          int64  
 2   cmp_fname_c1  float64
 3   cmp_fname_c2  float64
 4   cmp_lname_c1  float64
 5   cmp_lname_c2  float64
 6   cmp_sex       int64  
 7   cmp_bd        float64
 8   cmp_bm        float64
 9   cmp_by        float64
 10  cmp_plz       float64
 11  is_match      bool   
dtypes: bool(1), float64(8), int64(3)
memory usage: 531.8 MB
```

```
df.isna().sum()
```

```
id_1                  0
id_2                  0
```

```
cmp_fname_c1       1007
cmp_fname_c2    5645434
cmp_lname_c1          0
cmp_lname_c2    5645434
cmp_sex               0
cmp_bd              795
cmp_bm              795
cmp_by              795
cmp_plz           12843
is_match              0
dtype: int64
```

```
# HANDLIG MISSING DATA
df.drop(["cmp_fname_c2","cmp_lname_c2"],axis = 1,inplace = True)
```

```
df.isna().sum()
```

```
id_1               0
id_2               0
cmp_fname_c1     1007
cmp_lname_c1        0
cmp_sex             0
cmp_bd            795
cmp_bm            795
cmp_by            795
cmp_plz         12843
is_match            0
dtype: int64
```

```
# Imputing with mean value
from sklearn.impute import SimpleImputer
si = SimpleImputer(missing_values=np.nan, strategy='mean')


columns_with_missing_values  = ["cmp_fname_c1"]

for col in columns_with_missing_values:
  df[[col]] = si.fit_transform(df[[col]])
```

```
df.isna().sum()
```

```
id_1               0
id_2               0
cmp_fname_c1       0
cmp_lname_c1       0
cmp_sex            0
cmp_bd           795
cmp_bm           795
cmp_by           795
cmp_plz        12843
is_match           0
dtype: int64
```

```
# Imputing with mode value
from sklearn.impute import SimpleImputer
si = SimpleImputer(missing_values=np.nan, strategy='most_frequent')


columns_with_missing_values  = ["cmp_plz"]

for col in columns_with_missing_values:
  df[[col]] = si.fit_transform(df[[col]])
```

```
df.isna().sum()
```

```
id_1             0
id_2             0
cmp_fname_c1     0
cmp_lname_c1     0
cmp_sex          0
cmp_bd         795
cmp_bm         795
cmp_by         795
cmp_plz          0
is_match         0
dtype: int64
```

```
df.dropna(inplace = True)
df.isna().sum()
```

```
id_1            0
id_2            0
cmp_fname_c1    0
cmp_lname_c1    0
cmp_sex         0
cmp_bd          0
cmp_bm          0
cmp_by          0
cmp_plz         0
is_match        0
dtype: int64
```

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
id_1	5748337.0	33323.022882	23659.785801	1.0	13296.000000	29122.000000	50277.000000	99980.0
id_2	5748337.0	66586.666629	23621.537113	6.0	50055.000000	70674.000000	86476.000000	100000.0
cmp_fname_c1	5748337.0	0.712976	0.388695	0.0	0.285714	1.000000	1.000000	1.0
cmp_lname_c1	5748337.0	0.315554	0.334186	0.0	0.100000	0.181818	0.428571	1.0
cmp_sex	5748337.0	0.954995	0.207315	0.0	1.000000	1.000000	1.000000	1.0
cmp_bd	5748337.0	0.224465	0.417230	0.0	0.000000	0.000000	0.000000	1.0
cmp_bm	5748337.0	0.488855	0.499876	0.0	0.000000	0.000000	1.000000	1.0
cmp_by	5748337.0	0.222749	0.416091	0.0	0.000000	0.000000	0.000000	1.0
cmp_plz	5748337.0	0.005516	0.074063	0.0	0.000000	0.000000	0.000000	1.0

```
df
```

	id_1	id_2	cmp_fname_c1	cmp_lname_c1	cmp_sex	cmp_bd	cmp_bm	cmp_by	cmp_plz	is_match
0	37291	53113	0.833333	1.000000	1	1.0	1.0	1.0	0.0	True
1	39086	47614	1.000000	1.000000	1	1.0	1.0	1.0	1.0	True
2	70031	70237	1.000000	1.000000	1	1.0	1.0	1.0	1.0	True
3	84795	97439	1.000000	1.000000	1	1.0	1.0	1.0	1.0	True
4	36950	42116	1.000000	1.000000	1	1.0	1.0	1.0	1.0	True
...	...	...	...	...	...	...	...	...	...	...
574908	32517	73116	1.000000	0.222222	1	0.0	1.0	0.0	0.0	False
574909	67707	83757	0.111111	1.000000	1	0.0	0.0	0.0	0.0	False
574910	53258	91808	1.000000	0.000000	1	0.0	0.0	1.0	0.0	False
574911	31865	85285	1.000000	0.111111	1	0.0	1.0	0.0	0.0	False
574912	33119	76399	1.000000	0.000000	1	0.0	1.0	0.0	0.0	False

5748337 rows × 10 columns

```
# GETTING  TARGETS ATTRIBUTE
target = df["is_match"]

df.drop(["is_match"],axis = 1,inplace = True)
target
```

```
0          True
1          True
2          True
3          True
4          True
          ...
574908    False
574909    False
574910    False
574911    False
574912    False
Name: is_match, Length: 5748337, dtype: bool
```

```
target = target.replace({True: 1, False: 0})
```

```
target
```

```
0         1
1         1
```

```
2         1
3         1
4         1
         ..
574908    0
574909    0
574910    0
574911    0
574912    0
Name: is_match, Length: 5748337, dtype: int64
```

```
# FEATURES MODELLING
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(df,target, test_size=0.25, random_state=6)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(4311252, 9)
(1437085, 9)
(4311252,)
(1437085,)
```

```
# SCALING TO SETTING ALL ATTRIBUTES UNITS SIMILAR
from sklearn.preprocessing import StandardScaler,RobustScaler

#apply scaling
scaler = StandardScaler()


# x_train.iloc[:,1:] = scaler.fit_transform(x_train.iloc[:,1:])
# x_test.iloc[:,1:] = scaler.transform(x_test.iloc[:,1:])

x_train.iloc[:,0:] = scaler.fit_transform(x_train.iloc[:,0:])
x_test.iloc[:,0:] = scaler.transform(x_test.iloc[:,0:])

print(x_train.head(10))
```

```
              id_1      id_2  cmp_fname_c1  cmp_lname_c1   cmp_sex    cmp_bd  \
425475  0.950075  0.436041     -1.513068      2.048361  0.217053 -0.538092
113673 -0.720677 -1.194770      0.738308     -0.944199  0.217053 -0.538092
186340  0.206565  0.918010      0.738308     -0.570129  0.217053 -0.538092
10829  -0.822540 -1.174193     -1.834693      1.050841  0.217053 -0.538092
170072 -1.388531 -2.256813     -1.262915      0.053321  0.217053 -0.538092
9532   -1.090974  0.518775      0.738308     -0.023412  0.217053  1.858419
517330 -1.288951 -1.190494      0.738308     -0.944199  0.217053 -0.538092
178173  0.743942  0.765666      0.738308     -0.570129  0.217053 -0.538092
388770 -0.256253 -1.145951      0.738308     -0.611693  0.217053  1.858419
236559 -0.247419  1.266435      0.738308     -0.400097  0.217053  1.858419

          cmp_bm    cmp_by  cmp_plz
425475 -0.978137  1.868471  -0.0746
113673  1.022352 -0.535197  -0.0746
186340  1.022352 -0.535197  -0.0746
10829  -0.978137 -0.535197  -0.0746
170072 -0.978137 -0.535197  -0.0746
9532   -0.978137 -0.535197  -0.0746
517330 -0.978137  1.868471  -0.0746
178173  1.022352 -0.535197  -0.0746
388770 -0.978137 -0.535197  -0.0746
236559 -0.978137 -0.535197  -0.0746
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB


# Define a list of classifiers
classifiers = [
    DecisionTreeClassifier(),
    GaussianNB(),
    RandomForestClassifier(),
    LogisticRegression(),
    SVC()
]
```

```
# Lists to store evaluation metrics
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []

# Train and evaluate each classifier
for classifier in classifiers:
    classifier.fit(x_train, y_train)
    y_pred = classifier.predict(x_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    accuracy_scores.append(accuracy)
    precision_scores.append(precision)
    recall_scores.append(recall)
    f1_scores.append(f1)

    print(f"Classifier: {type(classifier)}")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")
    print("--------------------------------------------")
```

```
Classifier: <class 'sklearn.tree._classes.DecisionTreeClassifier'>
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000
--------------------------------------------
Classifier: <class 'sklearn.naive_bayes.GaussianNB'>
Accuracy: 0.9998
Precision: 0.9998
Recall: 0.9998
F1-Score: 0.9998
--------------------------------------------
Classifier: <class 'sklearn.ensemble._forest.RandomForestClassifier'>
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000
--------------------------------------------
Classifier: <class 'sklearn.linear_model._logistic.LogisticRegression'>
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000
--------------------------------------------
Classifier: <class 'sklearn.svm._classes.SVC'>
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000
--------------------------------------------
```

```
# Plotting performance of different models
labels = [type(classifier) for classifier in classifiers]
bar_width = 0.1

x = np.arange(len(classifiers))
x1 = x - 1.5 * bar_width
x2 = x - 0.5 * bar_width
x3 = x + 0.5 * bar_width
x4 = x + 1.5 * bar_width

plt.figure(figsize=(10, 8))

plt.bar(x1, accuracy_scores, width=bar_width, label='Accuracy')
plt.bar(x2, precision_scores, width=bar_width, label='Precision')
plt.bar(x3, recall_scores, width=bar_width, label='Recall')
plt.bar(x4, f1_scores, width=bar_width, label='F1-Score')

plt.xlabel('Classifiers')
plt.ylabel('Score')
plt.title('Performance of Different Models')
plt.xticks(x, labels, rotation=90)
plt.legend()
plt.tight_layout()
plt.show()
```