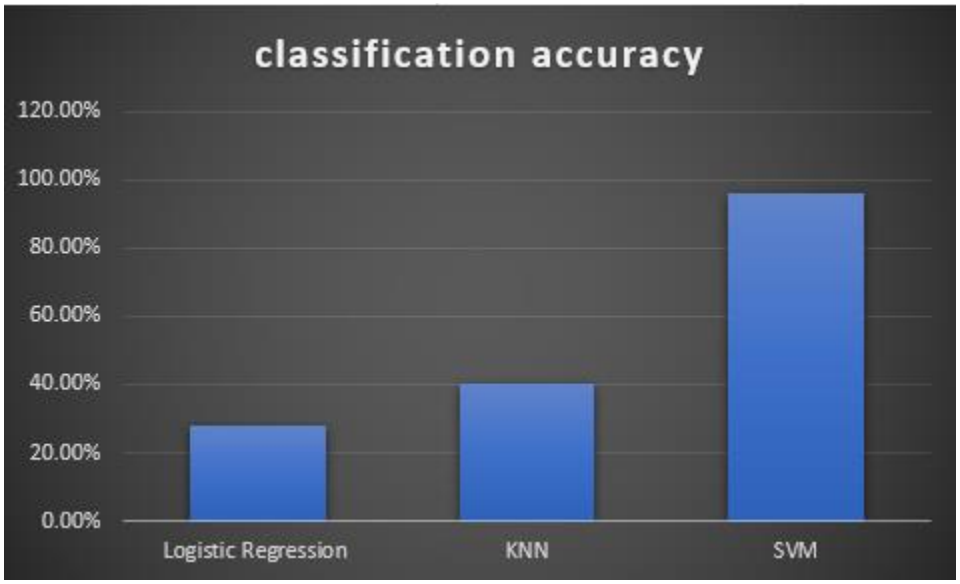
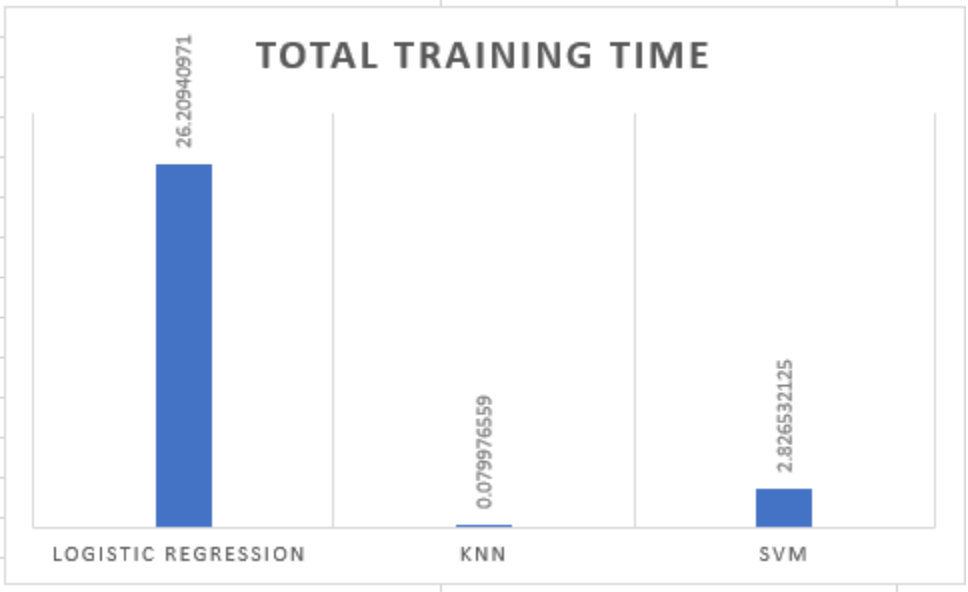
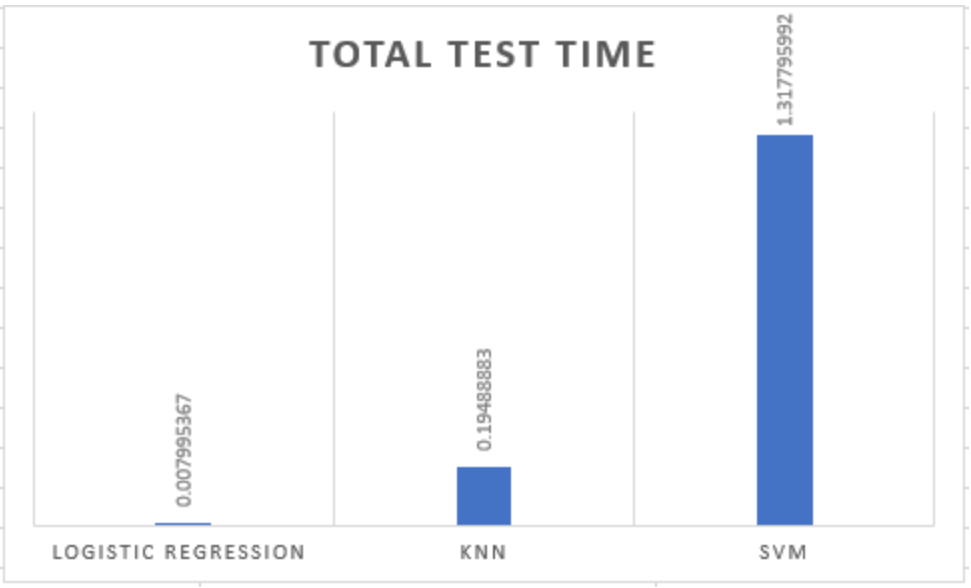
Milestone 2 Report

- Summarize the classification accuracy, total training time, and total test time using three bar graphs.







- we change some data preprocessing techniques to make data suitable for classification
  - Encode Categorical features

    We didn't use one hot encoding again to avoid curse of dimensionality!

```
def Encode_CatergoricalFeatures(data ,feature,lst):
    j=0
    for i in lst:
        data[feature] = data[feature].replace(i,j+1)
        j+=1
```

- Impute missing data

```
f = data["Rotten Tomatoes"].str.replace('%', ' ')
df = pd.DataFrame(f)
f = np.array(df)
f = f.astype(np.float)
f = df.fillna(np.nanmean(f)) # fill with mean of column.
data["Rotten Tomatoes"] = f.astype(np.float)

f = data["Age"].str.replace('all', '0')
f = f.str.replace('+', '')
df = pd.DataFrame(f)
f = np.array(df)
f = f.astype(np.float)
f = df.fillna(np.nanmean(f))

#f = df.fillna(f.mode().iloc[0])

data["Age"] = f.astype(np.int)


f = data["Runtime"]
df = pd.DataFrame(f)
f = np.array(df)
f = f.astype(np.float)
f = df.fillna(np.nanmean(f))
data["Runtime"] = f.astype(np.float)

f = data["Year"]
df = pd.DataFrame(f)
f = np.array(df)
data["Year"] = f.astype(np.float)
```

- Aggregate some features (Netflix-Hulu-Prime Video-Disney+)

```
data.rename(columns={"Netflix": "is_Streamed"}, inplace=True)


data["is_Streamed"] = data["is_Streamed"] + data["Hulu"] + data["Prime Video"] + data["Disney+"]
```

Then, make Feature Selection and drop very uncorrelated features

We got the correlation between features

	Year	Age	...	Genres	rate
Year	1.000000	0.107678	...	-0.137873	1.000000
Age	0.107678	1.000000	...	-0.129739	0.107678
Rotten Tomatoes	-0.037837	-0.056355	...	-0.007426	-0.037837
is_Streamed	0.020160	0.033630	...	-0.002835	0.020160
Genres	-0.137873	-0.129739	...	1.000000	-0.137873
rate	1.000000	0.107678	...	-0.137873	1.000000

We Selected the features by trial and error, In supervised learning, the trial and error method is mainly used, where we test to give us a known answer, We first enter values and compare them with the correct answer, so they would improve the results gradually until they give a precise answer.

```
Dropped_Cols = ["Title","Directors","Type", "Hulu", "Prime Video",
                "Disney+","Language","Country","Runtime"]
data.drop(Dropped_Cols, inplace=True, axis=1)
```

- At Last we Normalize and split data (80% training & 20% test)

```
scaler = StandardScaler()
data = scaler.fit_transform(data)
Y = Y.values.reshape(11743, )



data = np.array(data)
Y = np.array(Y)
shuffle(data, Y)



# 80% train , 20% test
x_train, x_test, y_train, y_test = train_test_split(
data, Y, test_size = 0.2, random_state = 42)
```

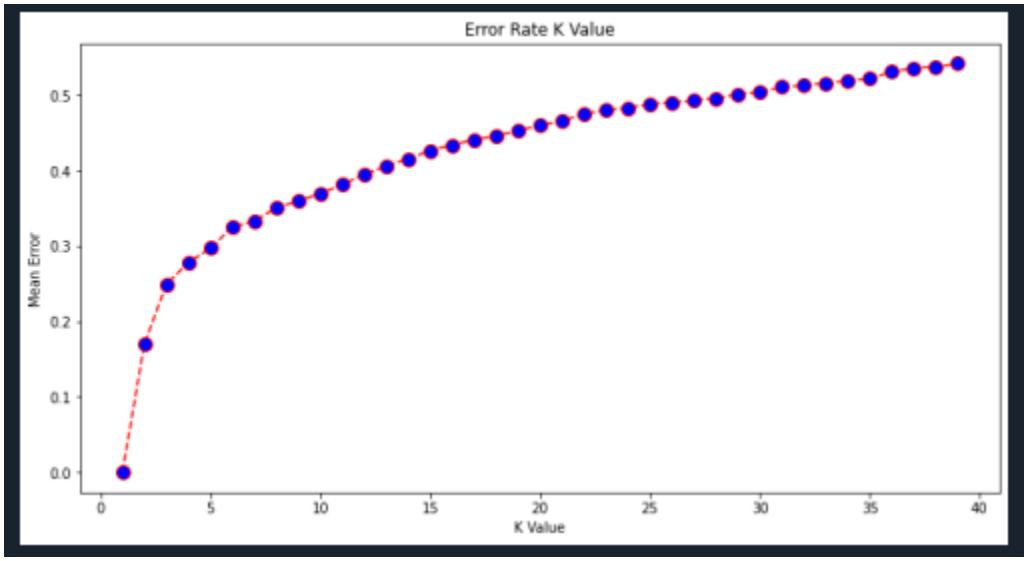
❖ For hyperparameter tuning we choose

- n_neighbors in KNeighborsClassifier

Calculating error for K values between 1 and 40 and plot this relation

```
# Calculating error for K values between 1 and 40
 for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn_start_time = time.time()
    knn.fit(X_train, y_train)
    knn_end_time = time.time()
    pred_i = knn.predict(X_train)
    error.append(np.mean(pred_i != y_train))
```

```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
        markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
plt.show()
```



We notice here increasing of K value Cause increasing in Mean Error!

The best K will be in range of 1 and 5 maybe 3

- C ,the regularized parameter in SVM

  Make C=1000
  SVC(C = 1000, kernel='rbf' ,gamma=0.1)
  Get : Training accuracy 100.0 %

  Make C=0.1
  SVC(C = 0.1, kernel='rbf' ,gamma=0.1)
  Get : Training accuracy is 11.0 %

  Make C=1 and kernel linear
  SVC(C = 1, kernel='linear' ,gamma=0.1)
  Get: Training accuracy is 84.0 %

Make C=1

SVC(C = 1, kernel='rbf' ,gamma=0.1)

Get: Training accuracy is 30.0 %

```
classifier = SVC(C = 1000, kernel='rbf' ,gamma=0.1)
SVM_start_time = time.time()
classifier.fit(X, y)
SVM_end_time = time.time()

SVM_training_time = SVM_end_time - SVM_start_time


print("total training time  ", SVM_training_time)
print("Training accuracy  ", round(classifier.score(X, y)*100), '%')


classifier = SVC(C = 0.1, kernel='rbf' ,gamma=0.1)
classifier.fit(X, y)
print('Training accuracy is', round(classifier.score(X, y)*100), '%')


classifier = SVC(C = 1, kernel='rbf' ,gamma=0.1)
classifier.fit(X, y)
print('Training accuracy is', round(classifier.score(X, y)*100), '%')


classifier = SVC(C = 1, kernel='linear' ,gamma=0.1)
classifier.fit(X, y)
print('Training accuracy is', round(classifier.score(X, y)*100), '%')
```

```
--------------------------SVM-----------------------


total training time   3.065234899520874
Training accuracy   100.0 %
Training accuracy is 11.0 %
Training accuracy is 30.0 %
Training accuracy is 84.0 %
Test accuracy   82.0 %
total test time  2.803386688232422

In [2]:
```

- Conclusion

We noticed that SVM have the test smallest error and the highest Accuracy as SVM is useful to solve any complex problem with a suitable Kernel,and it is efficient algorithm ,
it has a regularization parameter so causes generalization.
Knn is not very bad and easy to understand but Knn needs homogenous features,
Logistic Regression gives the worst accuracy as it relies on entire data and it's difficult to capture complex relationships using logistic regression.