```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
```

IMPORTING DATA

```
train_data = pd.read_csv("/content/train_data.txt",sep=':::', names=['ID', 'TITLE', 'GENRE', 'DESCRIPTION'])
train_data.head()
print(train_data.shape)
```

```
<ipython-input-13-f10998ac6811>:1: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex
  train_data = pd.read_csv("/content/train_data.txt",sep=':::', names=['ID', 'TITLE', 'GENRE', 'DESCRIPTION'])
(54214, 4)
```

```
test_data = pd.read_csv("/content/test_data.txt",sep=':::', names=['ID', 'TITLE', 'GENRE', 'DESCRIPTION'])
test_data.head()
print(test_data.shape)
```

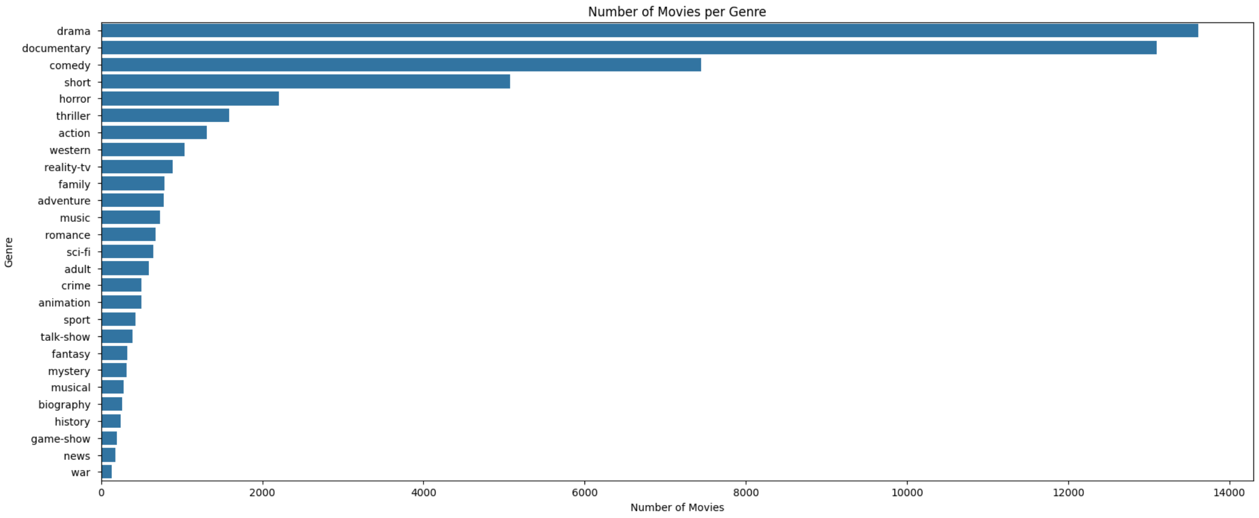
```
ling back to the 'python' engine because the 'c' engine does not support regex separators (separators > 1 char and different from '\
p=':::', names=['ID', 'TITLE', 'GENRE', 'DESCRIPTION'])
```

```
test_data_solution = pd.read_csv("/content/test_data_solution.txt",sep=':::',names=['ID', 'TITLE', 'GENRE', 'DESCRIPTION'])
test_data_solution.head()
print(test_data_solution.shape)
```

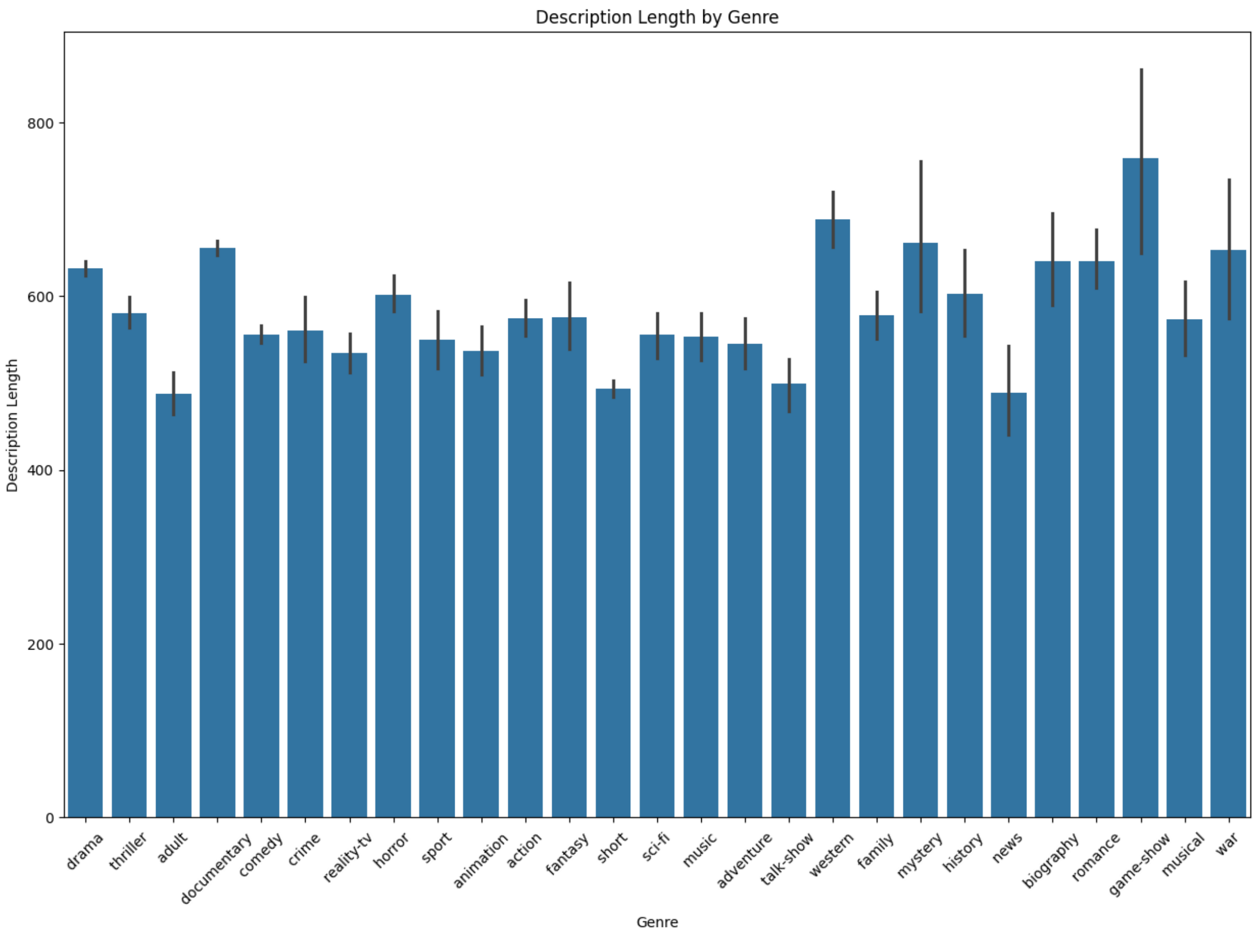
```
<ipython-input-20-224aace408ef>:1: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex
  test_data_solution = pd.read_csv("/content/test_data_solution.txt",sep=':::',names=['ID', 'TITLE', 'GENRE', 'DESCRIPTION'])
(54200, 4)
```

DATA VISUALIZATION METHOD

```
plt.figure(figsize=(20,8))
sns.countplot(y=train_data['GENRE'],order=train_data['GENRE'].value_counts().index)
plt.title('Number of Movies per Genre')
plt.xlabel('Number of Movies')
plt.ylabel('Genre')
plt.show()
```



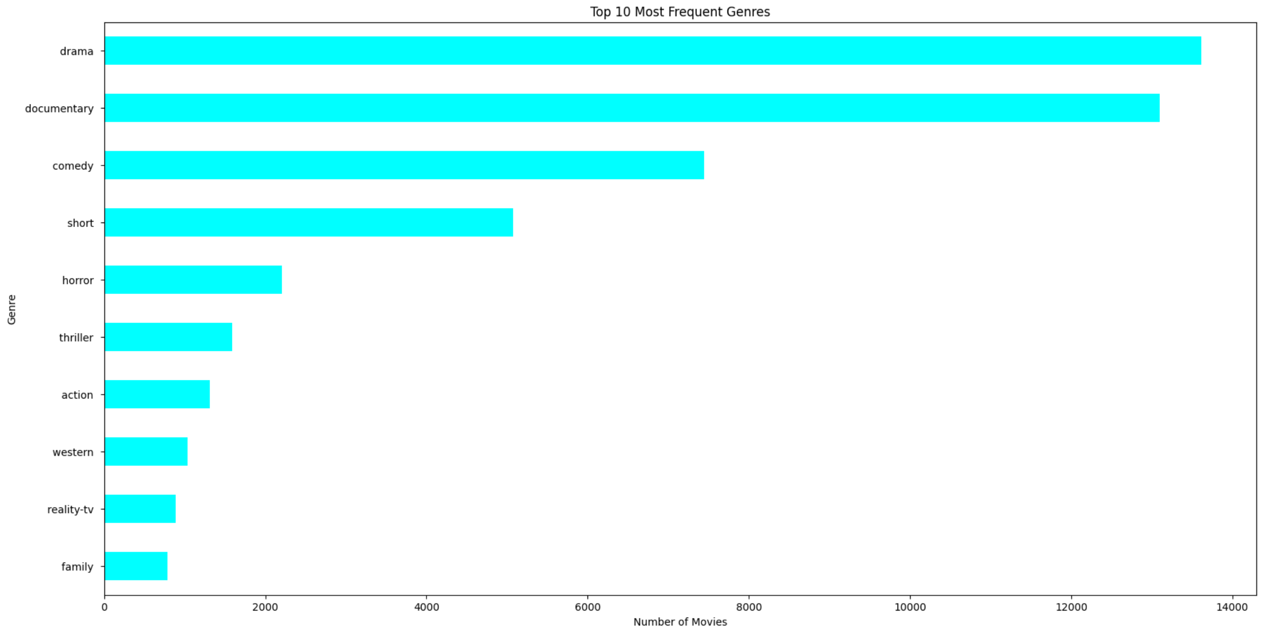
```
train_data['DESCRIPTION_length'] = train_data['DESCRIPTION'].apply(len)
plt.figure(figsize=(15, 10))
sns.barplot(x='GENRE', y='DESCRIPTION_length', data=train_data)
plt.title('Description Length by Genre')
plt.xticks(rotation=45)
plt.xlabel('Genre')
plt.ylabel('Description Length')
plt.show()
```



Now to get to known about top genre which mostly people watched

```
top_genres = train_data['GENRE'].value_counts().head(10)

plt.figure(figsize=(20, 10))
top_genres.plot(kind='barh', color='cyan')
plt.title('Top 10 Most Frequent Genres')
plt.xlabel('Number of Movies')
plt.ylabel('Genre')
plt.gca().invert_yaxis()  # Invert y-axis to have the genre with the most movies at the top
plt.show()
```



*now training and testing of the data *

```
train_data['DESCRIPTION'].fillna("", inplace=True)
test_data['DESCRIPTION'].fillna("", inplace=True)
```

```
<ipython-input-28-a8e7138b5873>:2: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise in a future er
  test_data['DESCRIPTION'].fillna("", inplace=True)
```

```
t_v = TfidfVectorizer(stop_words='english', max_features=100000)
X_train = t_v.fit_transform(train_data['DESCRIPTION'])
X_test = t_v.transform(test_data['DESCRIPTION'])
print(X_train)
print(X_test)
```

Show hidden output

```
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(train_data['GENRE'])
y_test = label_encoder.transform(test_data_solution['GENRE'])
```

```
[ 8 24  1 ...  7  5 12]
```

```
X_train_sub, X_val, y_train_sub, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

clf = LinearSVC()
clf.fit(X_train_sub, y_train_sub)

y_val_pred = clf.predict(X_val)
print("Validation Accuracy:", accuracy_score(y_val, y_val_pred))
print("Validation Classification Report:\n", classification_report(y_val, y_val_pred))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_classes.py:32: FutureWarning: The default value of `dual` will change from `Tr
  warnings.warn(
Validation Accuracy: 0.5836945494789265
Validation Classification Report:
               precision    recall  f1-score   support

           0       0.44      0.32      0.37       263
           1       0.74      0.44      0.55       112
           2       0.45      0.21      0.28       139
           3       0.47      0.15      0.23       104
           4       0.00      0.00      0.00        61
           5       0.53      0.59      0.56      1443
           6       0.39      0.07      0.11       107
           7       0.69      0.81      0.75      2659
           8       0.56      0.72      0.63      2697
           9       0.36      0.17      0.23       150
```

```
          10       0.13      0.03      0.04        74
          11       0.82      0.68      0.74        40
          12       0.00      0.00      0.00        45
          13       0.65      0.66      0.66       431
          14       0.61      0.53      0.57       144
          15       0.25      0.04      0.07        50
          16       0.43      0.05      0.10        56
          17       0.20      0.06      0.09        34
          18       0.49      0.25      0.33       192
          19       0.36      0.06      0.10       151
          20       0.50      0.28      0.36       143
          21       0.44      0.36      0.40      1045
          22       0.60      0.41      0.49        93
          23       0.62      0.25      0.35        81
          24       0.30      0.16      0.21       309
          25       0.50      0.05      0.09        20
          26       0.85      0.83      0.84       200

    accuracy                           0.58     10843
   macro avg       0.46      0.30      0.34     10843
weighted avg       0.56      0.58      0.56     10843
```

```
y_pred = clf.predict(X_test)
print("Test Accuracy:", accuracy_score(y_test, y_pred))
print("Test Classification Report:\n", classification_report(y_test, y_pred))
```

```
est Accuracy: 0.09357933579335793
est Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1314
           1       0.00      0.00      0.00       590
           2       0.00      0.00      0.00       775
           3       0.00      0.00      0.00       498
           4       0.00      0.00      0.00       264
           5       0.00      0.00      0.00      7446
           6       0.00      0.00      0.00       505
           7       0.00      0.00      0.00     13096
           8       0.00      0.00      0.00     13612
           9       0.00      0.00      0.00       783
          10       0.00      0.00      0.00       322
          11       0.00      0.00      0.00       193
          12       0.00      0.00      0.00       243
          13       0.00      0.00      0.00      2204
          14       0.00      0.00      0.00       731
          15       0.00      0.00      0.00       276
          16       0.00      0.00      0.00       318
          17       0.00      0.00      0.00       181
          18       0.00      0.00      0.00       883
          19       0.00      0.00      0.00       672
          20       0.00      0.00      0.00       646
          21       0.09      1.00      0.17      5072
          22       0.00      0.00      0.00       431
          23       0.00      0.00      0.00       391
          24       0.00      0.00      0.00      1590
          25       0.00      0.00      0.00       132
          26       0.00      0.00      0.00      1032

    accuracy                           0.09     54200
   macro avg       0.00      0.04      0.01     54200
eighted avg       0.01      0.09      0.02     54200

usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are il
 _warn_prf(average, modifier, msg_start, len(result))
usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are il
 _warn_prf(average, modifier, msg_start, len(result))
usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are il
 _warn_prf(average, modifier, msg_start, len(result))
```

```
from sklearn.naive_bayes import MultinomialNB
Mnb_classifier = MultinomialNB()
Mnb_classifier.fit(X_train, y_train)
```

```
▾ MultinomialNB
MultinomialNB()
```

```
Mnb_classifier.predict(X_test)
```

```
array([8, 8, 8, ..., 8, 8, 8])
```

```
from sklearn.linear_model import LogisticRegression
lr_classifier = LogisticRegression(max_iter=500)
lr_classifier.fit(X_train, y_train)
```

```
▾      LogisticRegression
LogisticRegression(max_iter=500)
```

```
lr_classifier.predict(X_test)
```

```
array([8, 8, 8, ..., 8, 8, 8])
```

Now designing a function show that we can predict the genre of the movie

```
def predict_movie(description):
    t_v1 = t_v.transform([description])
    pred_label = clf.predict(t_v1)
    return label_encoder.inverse_transform(pred_label)[0]

sample_descr_for_movie = "A movie where police cashes the criminal and shoot him"
print(predict_movie(sample_descr_for_movie))

sample_descr_for_movie1 = "A movie where person cashes a girl too get marry with him but girl refuses him."
print(predict_movie(sample_descr_for_movie1))
```

```
action
drama
```