IMDb top 5 Movies

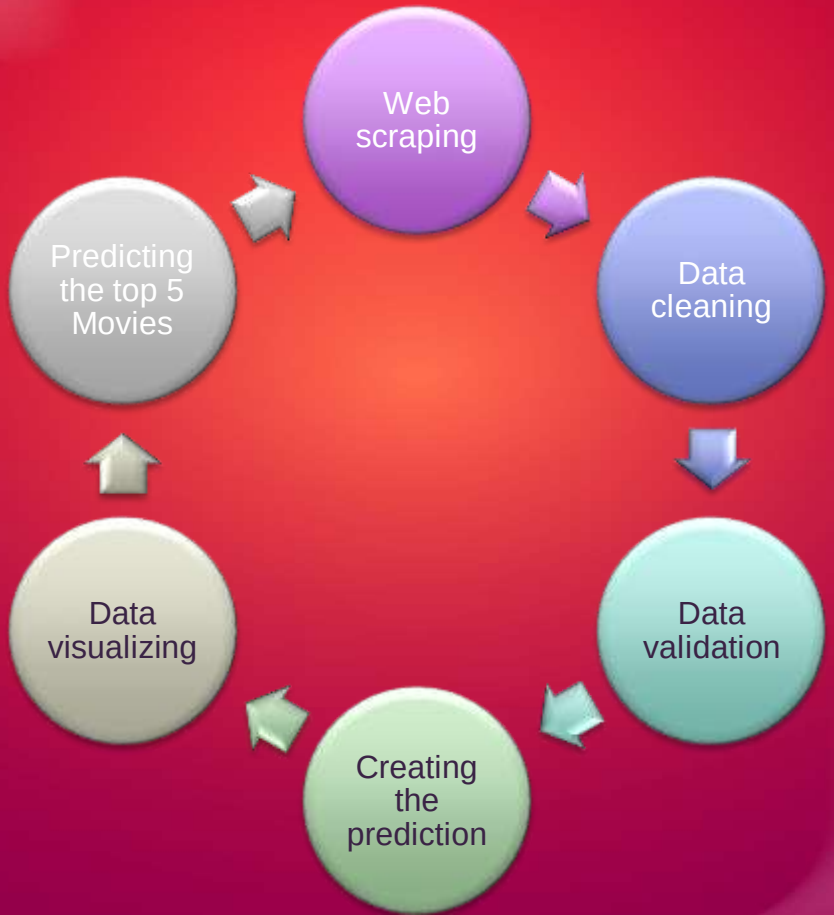
Done by : Rawan Al-Qahtani

Dr : Mejdal Al-Qahtani

introduction

1. In this project we have used the IMDB movies dataset that was tacking from the website through web-scraping with beautiful soap. This data is for a 1000 movie containing (name-year released - show length —IMDB rating —audience votes — movie type — metascore —US gross
2. This data was used to create a module that can predict the IMDB rating from the other data(audience votes-year released - show length- metascore - US gross).

code sequence



Web scraping


```
mt_score =[]
votes = []
us_gross =[]
```

In [77]:

```
a=1
while a!=1001:
    url= f"https://www.imdb.com/search/title/?groups=top_1000&start={a}"+"&ref_=adv_nxt"
    response = requests.get(url)
    page = response.text
    soup = BeautifulSoup(page, "lxml")

    mov_div=soup.find_all('div', class_='lister-item mode-advanced')


    for container in mov_div:
        name=container.h3.a.text
        mov_names.append (name)

        year= container.h3.find('span',class_='lister-item-year').text
        years.append(year)

        my_type =container.p.find('span',class_='genre').text if container.p.find('span',class_='genre').text else '-'
```

IMDB webpage was scraped to get dataset consist of 1000 movie with 8 categories for each.
 this data was sorted into data-Frame

```
movies.head(1000)
```

Out[78]:

	movie	years	type	IMDB_RATE	timeMin	metascore	votes	US_grossM
0	Dune: Part One	(2021)	\nAction, Adventure, Drama	8.2	155 min	74	371,789	-
1	Home Alone	(1990)	\nComedy, Family	7.6	103 min	63	519,525	$285.76M
2	Jai Bhim	(2021)	\nCrime, Drama	9.5	164 min	-	152,430	-
3	Ghostbusters	(1984)	\nAction, Comedy, Fantasy	7.8	105 min	71	385,920	$238.63M

Data cleaning

cleaning the numeric data from object to int:

```
In [79]: 1 print(movies.dtypes)

movie          object
years          object
type           object
IMDB_RATE     float64
timeMin        object
metascore      object
votes          object
US_grossM      object
dtype: object
```

```
[81]: 1 print(movies.dtypes)

movie          object
years           int32
type           object
IMDB_RATE     float64
timeMin         int32
metascore     float64
votes           int32
US_grossM     float64
dtype: object
```

converting the nan to 0 so it can be recognized as numeric data:

	years	IMDB_RATE	timeMin	metascore	votes	US_grossM
0	2021	8.2	155	74.0	371789	NaN
1	1990	7.6	103	63.0	519525	285.76
2	2021	9.5	164	NaN	152430	NaN
3	1984	7.8	105	71.0	385920	238.63
4	2001	7.6	152	65.0	713711	317.58
5	1999	8.7	136	73.0	1784082	171.48

	years	IMDB_RATE	timeMin	metascore	votes	US_grossM
0	2021	8.2	155	74.0	371789	0.00
1	1990	7.6	103	63.0	519525	285.76
2	2021	9.5	164	0.0	152430	0.00
3	1984	7.8	105	71.0	385920	238.63
4	2001	7.6	152	65.0	713711	317.58
5	1999	8.7	136	73.0	1784082	171.48

Data cleaning

converting the movies type into numarical variable

]:
```
1  type_dummies = pd.get_dummies(movies['type'],drop_first=True)
2
3  type_dummies.head()
```

]:

	Action, Adventure, Biography	Action, Adventure, Comedy	Action, Adventure, Crime	Action, Adventure, Drama	Action, Adventure, Family	Action, Adventure, Fantasy	Action, Adventure, Horror
0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0

convert the categorical data
to numerical

Data validation

Cross-Validation

[27]:
```
from sklearn.model_selection import KFold

X, y = Moviees.drop('IMDB_RATE',axis=1), Moviees['IMDB_RATE']

X, X_test, y, y_test = train_test_split(X, y, test_size=.2, random_state=10) #hold out 20% of the data for final testing

#this helps with the way kf will generate indices below
X, y = np.array(X), np.array(y)
```

[28]:
```
#run the CV

kf = KFold(n_splits=5, shuffle=True, random_state = 71)
cv_lm_r2s, cv_lm_reg_r2s = [], [] #collect the validation results for both models

for train_ind, val_ind in kf.split(X,y):

    X_train, y_train = X[train_ind], y[train_ind]
```

To ensures that the ML model picks up the right (relevant) patterns from the dataset

Data validation

K-fold

In [30]:
```
from sklearn.model_selection import cross_val_score
lm1 = LinearRegression()

cross_val_score(lm1, X, y, # estimator, features, target
                cv=5, # number of folds
                scoring='r2') # scoring metric
```

Out[30]:
```
array([0.46886132, 0.62093856, 0.48275947, 0.62971057, 0.32366894])
```

In [31]:
```
kf = KFold(n_splits=5, shuffle=True, random_state = 70)

print(np.mean(cross_val_score(lm1, X, y, cv=kf, scoring='r2')))
print(np.mean(cross_val_score(lm_reg, X, y, cv=kf, scoring='r2')))
```

```
0.5415672582519152
0.541567339165746
```

The low scores are accepted in this module since one of its input is audience votes which is human nature that can’ t behave in a predictable way or with certain pattern

Creating the prediction module

Data Regression

In [19]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge #ordinary linear regression + w/ ridge regularization
from sklearn.preprocessing import StandardScaler, PolynomialFeatures

X, y = Moviees.drop('IMDB_RATE',axis=1), Moviees['IMDB_RATE']


X, X_test, y, y_test = train_test_split(X, y, test_size=.2, random_state=10)
```

In [20]:
```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=.25, random_state=10)
```

In [21]:
```
#set up the 3 models we're choosing from:

lm = LinearRegression()

#Feature scaling for train, val, and test so that we can run our ridge model on each
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train.values)
```

Splitting the data to training_set 80% and testing_set 20%

Creating the prediction module

```
19 X_test_poly = poly.transform(X_test.values)
20
21 lm_poly = LinearRegression()
```

In [22]:

```
1 lm.fit(X_train, y_train)
2 print(f'Linear Regression val R^2: {lm.score(X_val, y_val):.3f}')
3
4 lm_reg.fit(X_train_scaled, y_train)
5 print(f'Ridge Regression val R^2: {lm_reg.score(X_val_scaled, y_val):.3f}')
6
7 lm_poly.fit(X_train_poly, y_train)
8 print(f'Degree 2 polynomial regression val R^2: {lm_poly.score(X_val_poly, y_val):.3f}')
```

```
Linear Regression val R^2: 0.580
Ridge Regression val R^2: 0.580
Degree 2 polynomial regression val R^2: 0.586
```

The training score

In [23]:

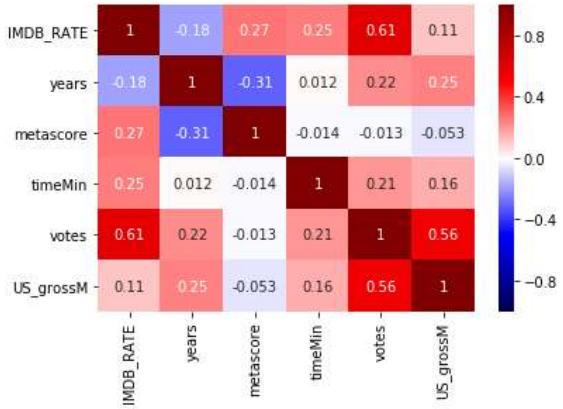
```
1 lm.fit(X,y)
2 print(f'Linear Regression test R^2: {lm.score(X_test, y_test):.3f}')
```

```
Linear Regression test R^2: 0.623
```

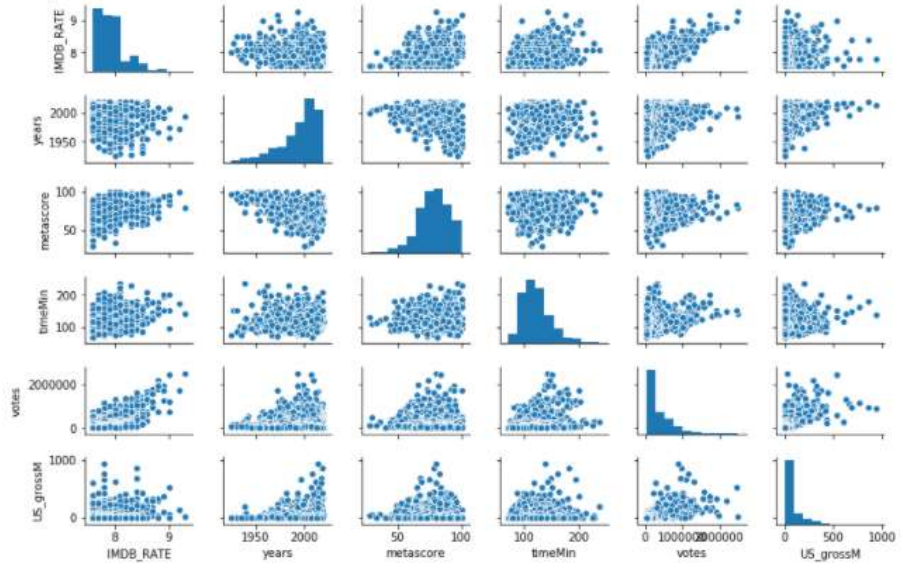
The testing score

Data visualizing

```
:  1  # for a better corr matrix
   2  sns.heatmap(MOVIE.corr(), cmap="seismic", annot=True, vmin=-1, vmax=1);
```

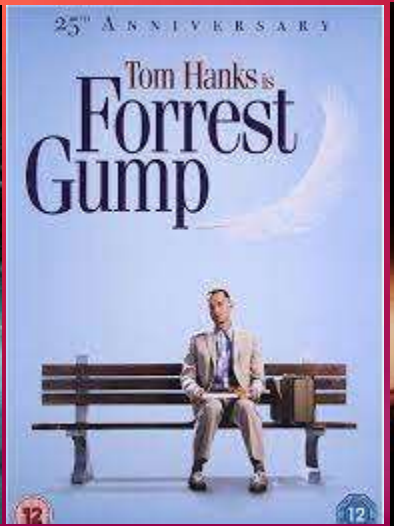


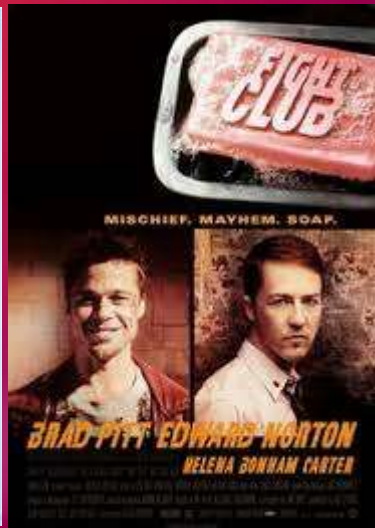

```
[36]:  1  # Plot all of the variable-to-variable relations as scatterplots
       2  sns.pairplot(MOVIE, height=1.2, aspect=1.5);
```



Predicting the IMDb rate to top 5 Movies

	movie	years	type	IMDB_RATE	timeMin	metascore	votes	US_grossM
8	The Shawshank Redemption	1994	\nDrama	?	142	80.0	2501001	28.34
21	The Dark Knight	2008	\nAction, Crime, Drama		152	84.0	2450916	534.86
23	Inception	2010	\nAction, Adventure, Sci-Fi		148	74.0	2199289	292.58
33	Fight Club	1999	\nDrama		139	66.0	1967527	37.03
53	Forrest Gump	1994	\nDrama, Romance		142	82.0	1930469	330.25





Predicting the IMDb rate to top 5 Movies

The real data

	movie	years	type	IMDB_RATE	timeMin	metascore	votes	US_grossM
8	The Shawshank Redemption	1994	\nDrama	9.3	142	80.0	2501001	28.34
21	The Dark Knight	2008	\nAction, Crime, Drama	9.0	152	84.0	2450916	534.86
23	Inception	2010	\nAction, Adventure, Sci-Fi	8.8	148	74.0	2199289	292.58
33	Fight Club	1999	\nDrama	8.8	139	66.0	1967527	37.03
53	Forrest Gump	1994	\nDrama, Romance	8.8	142	82.0	1930469	330.25

The input data to the prediction module

	years	timeMn	metascore	votes	US_grossM
0	1994	142	80.0	2500703	28.34
1	2008	152	84.0	2450683	534.86
2	2010	148	74.0	2199071	292.58
3	1999	139	66.0	1967340	37.03
4	1994	142	82.0	1930302	330.25

Comparing the output with the real data

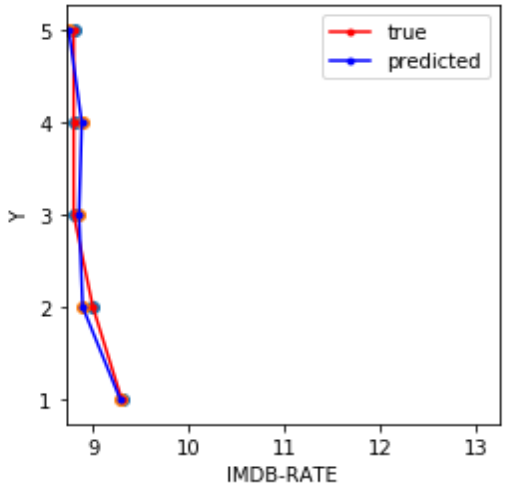
The output data

```
1  lm.predict(top_5.head())
```

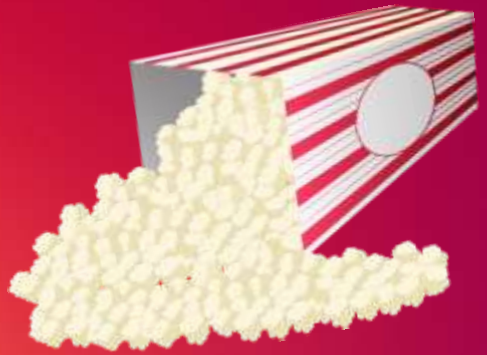
```
array([9.28156808, 8.89063007, 8.85285882, 8.87675444, 8.74433524])
```

Predicting the IMDb rate to top 5 Movies

```
Out[124]: <matplotlib.legend.Legend at 0x1bc0370e908>
```



Comparing the output with the real data



THANKS!