Data Cleaning - Feature Engineering

In [1]:

```
# Import reqd. libs.
import pandas as pd
import seaborn as sb
import numpy as np
import matplotlib.pyplot as plt
import os
%matplotlib inline
```

In [2]:

```
# Import the given data sets (personally despised and hence the dataset names)
hateset1 = pd.read_csv("bollywood.csv")
hateset2 = pd.read_csv("bollywood - 2.csv")
```

In [3]:

```
# Peek at the data
display(hateset1.head(3), hateset2.tail(3))
```

	No	RelDate	MovieName	ReleaseTime	Genre	Budget	BoxOfficeCollection	Youtı
0	1	Apr-18-2014	2 States	3	RomanceGenre	36Cr	104	
1	2	Jan-4-2013	Table No. 21	4	Thriller	Cr10	12	
2	3	Jul-18-2014	Amit Sahni Ki List	4	ComedyGenre	10Cr	4	

	Movie_ID	Release_Date	Movie_Name	Release_Time	Genre	BudgetCr	BoxOfficeColle
96	147	20-Mar-15	Dilliwali Zaalim Girlfriend	N	Comedy	32	
97	148	20-Mar-15	Hunterrr	N	Comedy	5	
98	149	23-May-14	Kochadaiiyaan	HS	Action	150	

In [ ]:

```

```

Clean the data

In [4]:

```
# Column names between data-sets differ
# Rename columns to a common name between the sets
# Hateset2 has a missing column TotalVotes (YoutubeLikeDislikes)
print('Hateset1 column names before rename:')
print(list(hateset1.columns))
print()
hateset1.rename(columns={'No':'MovieIndex',
                         'RelDate':'DateReleased',
                         'MovieName':'Name',
                         'ReleaseTime':'TimeReleased',
                         'Genre':'Genre',
                         'Budget':'Budget',
                         'BoxOfficeCollection':'Collection',
                         'YoutubeViews':'Views',
                         'YoutubeLikes':'Likes',
                         'YoutubeDislikes':'Dislikes',
                         'YoutubeLikeDislikes':'TotalVotes'},inplace=True)
print('Hateset1 column names after rename:')
print(list(hateset1.columns))
```

```
Hateset1 column names before rename:
['No', 'RelDate', 'MovieName', 'ReleaseTime', 'Genre', 'Budget', 'BoxOffic
eCollection', 'YoutubeViews', 'YoutubeLikes', 'YoutubeDislikes', 'YoutubeL
ikeDislikes']

Hateset1 column names after rename:
['MovieIndex', 'DateReleased', 'Name', 'TimeReleased', 'Genre', 'Budget',
'Collection', 'Views', 'Likes', 'Dislikes', 'TotalVotes']
```

In [5]:

```
print('Hateset2 column names before rename:')
print(list(hateset2.columns))
print()
hateset2.rename(columns={'Movie_ID':'MovieIndex',
                         'Release_Date':'DateReleased',
                         'Movie_Name':'Name',
                         'Release_Time':'TimeReleased',
                         'Genre':'Genre',
                         'BudgetCr':'Budget',
                         'BoxOfficeCollectionCr':'Collection',
                         'Youtube_Views':'Views',
                         'Youtube_Likes':'Likes',
                         'Youtube_Dislikes':'Dislikes'},inplace=True)
print('Hateset2 column names after rename:')
print(list(hateset2.columns))
```

```
Hateset2 column names before rename:
['Movie_ID', 'Release_Date', 'Movie_Name', 'Release_Time', 'Genre', 'Budge
tCr', 'BoxOfficeCollectionCr', 'Youtube_Views', 'Youtube_Likes', 'Youtube_
Dislikes']

Hateset2 column names after rename:
['MovieIndex', 'DateReleased', 'Name', 'TimeReleased', 'Genre', 'Budget',
'Collection', 'Views', 'Likes', 'Dislikes']
```

In [6]:

```
# Add missing column in second hateset
hateset2['TotalVotes'] = hateset2.Likes + hateset2.Dislikes
print('Hateset2 columns after adding missing column:')
print(list(hateset2.columns))
```

```
Hateset2 columns after adding missing column:
['MovieIndex', 'DateReleased', 'Name', 'TimeReleased', 'Genre', 'Budget',
'Collection', 'Views', 'Likes', 'Dislikes', 'TotalVotes']
```

In [7]:

```
# Check and correct null/na columns to avoid future errors
# Check and replace null/na to most sensible values
hateset1[hateset1.isna().any(axis=1)]
```

Out[7]:

	MovieIndex	DateReleased	Name	TimeReleased	Genre	Budget	Collection
9	10	Jan-11-2013	Gangoobai	4	DramaGenre	2Cr	Cr0.01
31	32	Feb-15-2013	Murder 3	1	Thriller	12Cr	25
39	40	Jun-20-2014	Humshakals	2	ComedyGenre	75Cr	63.7
46	47	Mar-15-2013	Mere Dad Ki Maruti	4	ComedyGenre	5Cr	11

In [8]:

```
# Replace Likes and Dislikes with correct values
index = hateset1[pd.isna(hateset1.Likes)].index
hateset1.loc[index,('Likes')] = hateset1.TotalVotes[index] - hateset1.Dislikes[index]

index = hateset1[pd.isna(hateset1.Dislikes)].index
hateset1.loc[index,('Dislikes')] = hateset1.TotalVotes[index] - hateset1.Likes[index]
hateset1[hateset1.isna().any(axis=1)]
```

Out[8]:

	MovieIndex	DateReleased	Name	TimeReleased	Genre	Budget	Collection	Views	Likes

In [9]:

```
hateset2[hateset2.isnull().any(axis=1)]
# Nothing here
```

Out[9]:

	MovieIndex	DateReleased	Name	TimeReleased	Genre	Budget	Collection	Views	Likes

In [10]:

```
# Columns types between data-sets differ
# Convert types between the sets to most sensible common type
display(hateset1.sort_index(axis=1).dtypes, hateset2.sort_index(axis=1).dtypes)
```

```
Budget           object
Collection       object
DateReleased     object
Dislikes        float64
Genre            object
Likes           float64
MovieIndex        int64
Name             object
TimeReleased      int64
TotalVotes        int64
Views             int64
dtype: object

Budget            int64
Collection      float64
DateReleased     object
Dislikes          int64
Genre            object
Likes             int64
MovieIndex        int64
Name             object
TimeReleased     object
TotalVotes        int64
Views             int64
dtype: object
```

In [11]:

```
# Likes and Dislikes to be int
hateset1.Likes = hateset1.Likes.astype('int64', copy=False)
hateset1.Dislikes = hateset1.Dislikes.astype('int64', copy=False)

# Name, Genre to be string in both hatesets
hateset1.Name = hateset1.Name.astype('string', copy=False)
hateset2.Name = hateset2.Name.astype('string', copy=False)
```

In [12]:

```
# Genre data is messed with string Genre and whitespaces
print('Before cleaning: ')
print('Hateset1: ' + str(set(hateset1.Genre)))
print('Hateset2: ' + str(set(hateset2.Genre)))
print()

hateset1.Genre = hateset1.Genre.str.replace('Genre','')
hateset1.Genre = hateset2.Genre.str.strip().astype('string', copy=False)
hateset2.Genre = hateset2.Genre.str.strip().astype('string', copy=False)

print('After cleaning: ')
print('Hateset1: ' + str(set(hateset1.Genre)))
print('Hateset2: ' + str(set(hateset2.Genre)))
```

```
Before cleaning:
Hateset1: {' DramaGenre', 'RomanceGenre', ' Drama ', 'ActionGenre', 'Comed
yGenre', 'Thriller ', 'Action ', 'Thriller'}
Hateset2: {'Romance', ' Drama ', 'Action', 'Thriller', 'Comedy'}

After cleaning:
Hateset1: {'Romance', 'Action', 'Thriller', 'Drama', 'Comedy'}
Hateset2: {'Romance', 'Action', 'Thriller', 'Drama', 'Comedy'}
```

In [13]:

```
# Budget data is messed with string Cr
# Convert Budget to string to ease cleaning and make it as int
hateset1.Budget.astype('string')
hateset1.Budget = hateset1.Budget.str.replace('Cr','')
hateset1.Budget = hateset1.Budget.astype('int64', copy=False)
```

In [14]:

```
# Collection data is messed with string Cr
# Convert Collection to string to ease cleaning and make it as int
hateset1.Collection.astype('string')
hateset1.Collection = hateset1.Collection.str.replace('Cr','')
hateset1.Collection = hateset1.Collection.astype('float', copy=False)
```

In [15]:

```
# Clean the DateReleased column and data
hateset1[pd.to_datetime(hateset1.DateReleased, errors='coerce', infer_datetime_format=T
rue).isnull()]
```

Out[15]:

	MovieIndex	DateReleased	Name	TimeReleased	Genre	Budget	Collection	Vie
6	7	Sqp-19-2014	Daawat-E-Ishq	4	Drama	30	24.60	39050
12	13	Jum-27-2014	Ek Villain	2	Thriller	35	105.50	45187
17	18	juu-6-2014	Filmistaan	2	Romance	10	6.00	9890
34	35	Feb-21-204	Highway	4	Comedy	30	27.25	30430

In [16]:

```
# Getting the actual dates from an external KB (google.com)
hateset1.loc[6, ['DateReleased']] = 'Sep-19-2014'
hateset1.loc[12, ['DateReleased']] = 'Jun-27-2014'
hateset1.loc[17, ['DateReleased']] = 'Jun-6-2014'
hateset1.loc[34, ['DateReleased']] = 'Feb-21-2014'
hateset1.DateReleased = pd.to_datetime(hateset1.DateReleased, errors='coerce', infer_da
tetime_format=True)
```

In [17]:

```
hateset2[pd.to_datetime(hateset2.DateReleased, errors='coerce', infer_datetime_format=T
rue).isnull()]
```

Out[17]:

	MovieIndex	DateReleased	Name	TimeReleased	Genre	Budget	Collection	Views	Likes

In [18]:

```
hateset2.DateReleased = pd.to_datetime(hateset2.DateReleased, errors='coerce', infer_da
tetime_format=True)
```

In [19]:

```
# Check that we are satisfied with the cleaning and data types
display(hateset1.sort_index(axis=1).dtypes, hateset2.sort_index(axis=1).dtypes)
```

```
Budget                  int64
Collection            float64
DateReleased    datetime64[ns]
Dislikes                int64
Genre                  string
Likes                   int64
MovieIndex              int64
Name                   string
TimeReleased            int64
TotalVotes              int64
Views                   int64
dtype: object

Budget                  int64
Collection            float64
DateReleased    datetime64[ns]
Dislikes                int64
Genre                  string
Likes                   int64
MovieIndex              int64
Name                   string
TimeReleased           object
TotalVotes              int64
Views                   int64
dtype: object
```

In [20]:

```
# Merge the very hated data sets
# Note that TimeReleased is still a mixed data type column and we will clean this after merge
hateset = pd.concat([hateset1,hateset2], ignore_index=True)
display(hateset.head(2), hateset.tail(2))
```

	MovieIndex	DateReleased	Name	TimeReleased	Genre	Budget	Collection	Views
0	1	2014-04-18	2 States	3	Romance	36	104.0	8576361
1	2	2013-01-04	Table No. 21	4	Thriller	10	12.0	1087320

	MovieIndex	DateReleased	Name	TimeReleased	Genre	Budget	Collection	
148	148	2015-03-20	Hunterrr	N	Comedy	5	11.89	4
149	149	2014-05-23	Kochadaiiyaan	HS	Action	150	120.00	4

In [21]:

```
# Check resulting duplicates on the data-set's index which is MovieIndex
hateset.iloc[:,0].duplicated().sum()
```

Out[21]:

```
1
```

In [22]:

```
# Alternate way of testing uniques
hateset.MovieIndex.value_counts().head(2)
```

Out[22]:

```
51     2
149    1
Name: MovieIndex, dtype: int64
```

In [23]:

```
# We have one duplicate row, get and sort it out
hateset[hateset.MovieIndex == hateset[hateset.MovieIndex.duplicated()].index[0]]
```

Out[23]:

	MovieIndex	DateReleased	Name	TimeReleased	Genre	Budget	Collection	Viev
50	51	2014-07-07	Lekar Hum Deewana Dil	4	Action	16	2.5	39983
51	51	2014-07-04	Lekar Hum Deewana Dil	N	Romance	16	2.5	39983

In [24]:

```
# The first hate set was more prone to errors and hence we keep the second one
hateset.drop(index=hateset[(hateset.MovieIndex == 51) & (hateset.TimeReleased == 4)].in
dex, inplace=True, axis=0)
# Set the index to MovieIndex
hateset.set_index('MovieIndex', inplace=True)
```

In [25]:

```
# Clean TimeReleased data
print('Before cleaning: ')
print('TimeReleased: ' + str(set(hateset.TimeReleased)))
print()

def timeMap(l):
    mapper = {1:"FS", 2:"HS", 3:"LW", 4:"N", 44:"N", "LW":"LW", "N":"N", "FS":"FS", "H
S":"HS"}
    return mapper.get(l, 'NaN')
hateset.TimeReleased = hateset.TimeReleased.map(lambda l:timeMap(l))
hateset.TimeReleased = hateset.TimeReleased.astype('string', copy=False)

print('After cleaning: ')
print('TimeReleased: ' + str(set(hateset.TimeReleased)))
```

```
Before cleaning:
TimeReleased: {'N', 1, 2, 3, 4, 44, 'LW', 'HS', 'FS'}

After cleaning:
TimeReleased: {'N', 'HS', 'LW', 'FS'}
```

In [26]:

```
# Add ROI column to hateset
hateset['ROI'] = (hateset.Collection - hateset.Budget)/hateset.Budget
```

In [27]:

```
hateset.dtypes
```

Out[27]:

```
DateReleased    datetime64[ns]
Name                    string
TimeReleased            string
Genre                   string
Budget                   int64
Collection             float64
Views                    int64
Likes                    int64
Dislikes                 int64
TotalVotes               int64
ROI                    float64
dtype: object
```

In [ ]:

```

```

Q1. Identify and rectify the eight prominent data quality issues present in the given datasets.

In [28]:

```
### Below line items were performed in order listed from the top of the notebook
# 1.  Column indices/names
# 2.  Missing column
# 3.  Genre enumeration set cleaning
# 4.  Budget column cleaning
# 5.  Collection column cleaning
# 6.  DateReleased column cleaning
# 7.  Cleaning NaNs from Likes
# 8.  Cleaning NaNs from Dislikes
# 9.  Removing duplicated records
# 10. Cleaning TimeReleased enum
# 11. Set the column types for easy operations
```

Q2. How many records are present in the dataset? Print the metadata information of dataset.

In [29]:

```
print("""Hateset1 :  {} records
Hateset2 :  {} records
   Total : {} records""".format(hateset1.shape[0], hateset2.shape[0], hateset.shape[0
]))
# A duplicate record was removed and hence you will see one less in Total records
```

```
Hateset1 :  51 records
Hateset2 :  99 records
   Total : 149 records
```

In [30]:

```
# 149 records are present in the dataset
# Some more Metadata below. We have already seen the data-types and column listings
print("""Size - {}
Shape - {}
Dimensions - {}
""".format(hateset.size, hateset.shape, hateset.ndim))
print(hateset.info())
```

```
Size - 1639
Shape - (149, 11)
Dimensions - 2

<class 'pandas.core.frame.DataFrame'>
Int64Index: 149 entries, 1 to 149
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype         
---  ------        --------------  -----         
 0   DateReleased  149 non-null    datetime64[ns]
 1   Name          149 non-null    string        
 2   TimeReleased  149 non-null    string        
 3   Genre         149 non-null    string        
 4   Budget        149 non-null    int64         
 5   Collection    149 non-null    float64       
 6   Views         149 non-null    int64         
 7   Likes         149 non-null    int64         
 8   Dislikes      149 non-null    int64         
 9   TotalVotes    149 non-null    int64         
 10  ROI           149 non-null    float64       
dtypes: datetime64[ns](1), float64(2), int64(5), string(3)
memory usage: 14.0 KB
None
```

Q3. How many movies in each genre got released in different release times?

In [31]:

```
dict(hateset.Genre.value_counts())
```

Out[31]:

```
{'Comedy': 37, 'Drama': 34, 'Thriller': 32, 'Romance': 26, 'Action': 20}
```

In [32]:

```
# Comedy saw 37 movie releases while Drama had 34 releases, also 32 Thriller, 26 Romanc
e and 20 Action movies were released.
# Break-up of each Genre w.r.t the Release Time is shown below
hateset.groupby([hateset.Genre, hateset.TimeReleased]).Genre.count().unstack()
```

Out[32]:

TimeReleased	FS	HS	LW	N
Genre				
Action	3	1	3	13
Comedy	3	3	4	27
Drama	3	7	2	22
Romance	4	3	5	14
Thriller	4	4	1	23

Q4. Which month of the year, maximum movie releases are seen?

In [33]:

```
# Year is not specified, check how many years are present
set(hateset.DateReleased.dt.year.unique())
```

Out[33]:

```
{2013, 2014, 2015}
```

In [34]:

```
# We have 3 years, check which month of each year maximum movie releases are seen
display(hateset[hateset.DateReleased.dt.year == 2013].groupby(by=[hateset.DateReleased.
dt.year,
                                                                  hateset.DateReleased.
dt.strftime('%B')]).
        agg({'DateReleased':'count'}).DateReleased.sort_values(ascending=False).rename_
axis(["Year", "Month"]).head(),
        hateset[hateset.DateReleased.dt.year == 2014].groupby(by=[hateset.DateReleased.
dt.year,
                                                                  hateset.DateReleased.
dt.strftime('%B')]).
        agg({'DateReleased':'count'}).DateReleased.sort_values(ascending=False).rename_
axis(["Year", "Month"]).head(),
        hateset[hateset.DateReleased.dt.year == 2015].groupby(by=[hateset.DateReleased.
dt.year,hateset.
                                                                  DateReleased.dt.strft
ime('%B')]).
        agg({'DateReleased':'count'}).DateReleased.sort_values(ascending=False).rename_
axis(["Year", "Month"]).head())
```

```
Year  Month
2013  July       9
      March      8
      January    8
      October    6
      May        6
Name: DateReleased, dtype: int64


Year  Month
2014  May         12
      March        8
      July         7
      January      7
      February     6
Name: DateReleased, dtype: int64

Year  Month
2015  January     5
      February    4
      March       3
Name: DateReleased, dtype: int64
```

In [35]:

```
## We can see thus infer the following -
# 1) July has the maximum movie releases (9) in 2013
# 2) May has the maximum movie releases (12) in 2014
# 3) January has maximum movie releases (5) in 2015
```

In [36]:

```
# Aggregating over all years (since the question is ambiguous - which month of which ye
ar???)
print(hateset.groupby(by=[hateset.DateReleased.dt.strftime('%B')]).agg({'DateReleased':
'count'}).
      DateReleased.sort_values(ascending=False).rename_axis(["Month"]).head(3))
# January has the higest of 20 movie releases over 2013/14/15
```

```
Month
January    20
March      19
May        18
Name: DateReleased, dtype: int64
```

Q5. Which month of the year typically sees most releases of high budgeted movies, that is, movies with budget of 25 crore or more?

In [37]:

```
display(hateset[(hateset.Budget >= 25) & (hateset.DateReleased.dt.year == 2013)].
        groupby(by=[hateset.DateReleased.dt.year, hateset.DateReleased.dt.strftime('%B'
)]).
        agg({'DateReleased':'count'}).DateReleased.sort_values(ascending=False).rename_
axis(["Year", "Month"]).head(),
        hateset[(hateset.Budget >= 25) & (hateset.DateReleased.dt.year == 2014)].
        groupby(by=[hateset.DateReleased.dt.year, hateset.DateReleased.dt.strftime('%B'
)]).
        agg({'DateReleased':'count'}).DateReleased.sort_values(ascending=False).rename_
axis(["Year", "Month"]).head(),
        hateset[(hateset.Budget >= 25) & (hateset.DateReleased.dt.year == 2015)].
        groupby(by=[hateset.DateReleased.dt.year, hateset.DateReleased.dt.strftime('%B'
)]).
        agg({'DateReleased':'count'}).DateReleased.sort_values(ascending=False).rename_
axis(["Year", "Month"]))
```

```
Year  Month
2013  July         5
      September    4
      August       4
      November     3
      February     3
Name: DateReleased, dtype: int64

Year  Month
2014  March        4
      February     4
      April        4
      November     3
      June         3
Name: DateReleased, dtype: int64

Year  Month
2015  January      3
      February     2
      March        1
Name: DateReleased, dtype: int64
```

In [38]:

```
## We can conclude these facts about high budgeted movies -
# 1) 2013 - July had a highest of 5 releases
# 2) 2014 - February, March and April equally saw the highest of 4 releases each
# 3) 2015 - January had a highest of 3 releases
```

In [39]:

```
# Aggregating over all years (since the question is ambiguous - which month of which ye
ar???)
print(hateset[hateset.Budget >= 25].groupby(by=[hateset.DateReleased.dt.strftime('%B'
)]).agg({'DateReleased':'count'}).
      DateReleased.sort_values(ascending=False).rename_axis(["Month"]).head(3))
# February has the highest of 9 high budget movie releases over 2013/14/15
```

```
Month
February    9
January     8
March       7
Name: DateReleased, dtype: int64
```

Q6. Which are the top 10 movies with maximum return of investment (ROI)? Calculate ROI as (Box office collection – Budget) / Budget.

In [40]:

```
hateset.sort_values(by='ROI', ascending=False).loc[:,['Name','ROI']].head(10).style.for
mat({"ROI": "{:,.2f} Cr"}).hide_index()
```

Out[40]:

Name	ROI
Rajdhani Express	53.71 Cr
Aashiqui 2	8.17 Cr
PK	7.65 Cr
Grand Masti	7.51 Cr
The Lunchbox	7.50 Cr
Fukrey	6.24 Cr
Mary Kom	5.93 Cr
Shahid	5.67 Cr
Humpty Sharma Ki Dulhania	5.50 Cr
Bhaag Milkha Bhaag	4.47 Cr

In [41]:

```
# Top ten movies with their ROI values in Crores
```