Airline Satisfaction

1) Use the dataset named 'Airline Satisfaction.xlsx': It contains the data of 10000 airline customers and their details. Build a classification model to solve the below questions. Satisfaction is the target column a) What is the accuracy if the problem is solved using Random Forest model? b) What is the accuracy if the problem is solved using Support Vector Machine model?

Ans 1a

In [1]:
```
#Step1 import the data set
import pandas as pd
AS=pd.read_excel("Airline Satisfaction.xlsx")
AS
```

Out[1]:

	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	b
0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4	
1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	2	
2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	2	
3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	5	
4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	3	
...	...	...	...	...	...	...	...	...	...	
9995	124365	Male	Loyal Customer	50	Business travel	Business	3599	3	3	
9996	22044	Male	Loyal Customer	38	Business travel	Business	3873	5	5	
9997	14057	Female	Loyal Customer	39	Business travel	Business	319	4	4	
9998	113848	Male	Loyal Customer	52	Business travel	Business	1363	5	5	
9999	1865	Female	Loyal Customer	41	Business travel	Business	3938	4	4	

10000 rows × 24 columns

In [2]:
```
AS.isnull().sum()
```

Out[2]:
```
id                                  0
Gender                              0
Customer Type                       0
Age                                 0
Type of Travel                      0
Class                               0
Flight Distance                     0
Inflight wifi service               0
Departure/Arrival time convenient   0
Ease of Online booking              0
Gate location                       0
Food and drink                      0
Online boarding                     0
Seat comfort                        0
Inflight entertainment              0
On-board service                    0
Leg room service                    0
Baggage handling                    0
Checkin service                     0
Inflight service                    0
Cleanliness                         0
Departure Delay in Minutes          0
Arrival Delay in Minutes           26
satisfaction                        0
dtype: int64
```

In [3]:
```
AS.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 24 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   id                                 10000 non-null  int64
 1   Gender                             10000 non-null  object
 2   Customer Type                      10000 non-null  object
 3   Age                                10000 non-null  int64
 4   Type of Travel                     10000 non-null  object
 5   Class                              10000 non-null  object
 6   Flight Distance                    10000 non-null  int64
 7   Inflight wifi service              10000 non-null  int64
 8   Departure/Arrival time convenient  10000 non-null  int64
 9   Ease of Online booking             10000 non-null  int64
 10  Gate location                      10000 non-null  int64
 11  Food and drink                     10000 non-null  int64
 12  Online boarding                    10000 non-null  int64
 13  Seat comfort                       10000 non-null  int64
 14  Inflight entertainment             10000 non-null  int64
 15  On-board service                   10000 non-null  int64
 16  Leg room service                   10000 non-null  int64
 17  Baggage handling                   10000 non-null  int64
 18  Checkin service                    10000 non-null  int64
 19  Inflight service                   10000 non-null  int64
 20  Cleanliness                        10000 non-null  int64
 21  Departure Delay in Minutes         10000 non-null  int64
 22  Arrival Delay in Minutes           9974 non-null   float64
 23  satisfaction                       10000 non-null  int64
dtypes: float64(1), int64(19), object(4)
memory usage: 1.8+ MB
```

In [4]:
```
# We have 26 missing values in the field 'Arrival Delay in Minutes' these rows can
# to decide with which value to fill we check for skewness
AS['Arrival Delay in Minutes'].skew()
```

Out[4]:
```
7.681208890883187
```

In [5]:
```
# the skewness value is >1 therefore we can fillna with median
AS['Arrival Delay in Minutes'].fillna(AS['Arrival Delay in Minutes'].median(), inpl
```

In [6]:
```
AS.isnull().sum()
```

Out[6]:
```
id                                   0
Gender                               0
Customer Type                        0
Age                                  0
Type of Travel                       0
Class                                0
Flight Distance                      0
Inflight wifi service                0
Departure/Arrival time convenient    0
Ease of Online booking               0
Gate location                        0
Food and drink                       0
Online boarding                      0
Seat comfort                         0
Inflight entertainment               0
On-board service                     0
Leg room service                     0
Baggage handling                     0
Checkin service                      0
Inflight service                     0
Cleanliness                          0
Departure Delay in Minutes           0
Arrival Delay in Minutes             0
satisfaction                         0
dtype: int64
```

In [7]:
```
AS.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 24 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   id                                 10000 non-null  int64  
 1   Gender                             10000 non-null  object 
 2   Customer Type                      10000 non-null  object 
 3   Age                                10000 non-null  int64  
 4   Type of Travel                     10000 non-null  object 
 5   Class                              10000 non-null  object 
 6   Flight Distance                    10000 non-null  int64  
 7   Inflight wifi service              10000 non-null  int64  
 8   Departure/Arrival time convenient  10000 non-null  int64  
 9   Ease of Online booking             10000 non-null  int64  
 10  Gate location                      10000 non-null  int64  
 11  Food and drink                     10000 non-null  int64  
 12  Online boarding                    10000 non-null  int64  
 13  Seat comfort                       10000 non-null  int64  
 14  Inflight entertainment             10000 non-null  int64  
 15  On-board service                   10000 non-null  int64  
 16  Leg room service                   10000 non-null  int64  
 17  Baggage handling                   10000 non-null  int64  
 18  Checkin service                    10000 non-null  int64  
 19  Inflight service                   10000 non-null  int64  
 20  Cleanliness                        10000 non-null  int64  
 21  Departure Delay in Minutes         10000 non-null  int64  
 22  Arrival Delay in Minutes           10000 non-null  float64
 23  satisfaction                       10000 non-null  int64  
dtypes: float64(1), int64(19), object(4)
memory usage: 1.8+ MB
```

In [8]:

```
# we have 4 categorical variables, with no order of importance
# Applying Dummy variable (get_dummies) to convert the text columns into numerical

ASF= pd.get_dummies(AS, columns = ['Gender','Customer Type','Type of Travel','Class
ASF
```

Out[8]:

	id	Age	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	Food and drink	Online boarding	con
0	70172	13	460	3	4	3	1	5	3	
1	5047	25	235	3	2	3	3	1	3	
2	110028	26	1142	2	2	2	2	5	5	
3	24026	25	562	2	5	5	5	2	2	
4	119299	61	214	3	3	3	3	4	5	
...	...	...	...	...	...	...	...	...	...	
9995	124365	50	3599	3	3	3	3	4	5	
9996	22044	38	3873	5	5	5	5	5	5	
9997	14057	39	319	4	4	4	4	5	4	
9998	113848	52	1363	5	5	5	5	2	5	
9999	1865	41	3938	4	4	4	4	2	4	

10000 rows × 25 columns

In [9]:

```
ASF.info() # confirming that no null values & no string values in data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 25 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   id                                 10000 non-null  int64  
 1   Age                                10000 non-null  int64  
 2   Flight Distance                    10000 non-null  int64  
 3   Inflight wifi service              10000 non-null  int64  
 4   Departure/Arrival time convenient  10000 non-null  int64  
 5   Ease of Online booking             10000 non-null  int64  
 6   Gate location                      10000 non-null  int64  
 7   Food and drink                     10000 non-null  int64  
 8   Online boarding                    10000 non-null  int64  
 9   Seat comfort                       10000 non-null  int64  
 10  Inflight entertainment             10000 non-null  int64  
 11  On-board service                   10000 non-null  int64  
 12  Leg room service                   10000 non-null  int64  
 13  Baggage handling                   10000 non-null  int64  
 14  Checkin service                    10000 non-null  int64  
 15  Inflight service                   10000 non-null  int64  
 16  Cleanliness                        10000 non-null  int64  
 17  Departure Delay in Minutes         10000 non-null  int64  
 18  Arrival Delay in Minutes           10000 non-null  float64
 19  satisfaction                       10000 non-null  int64  
 20  Gender_Male                        10000 non-null  uint8  
 21  Customer Type_disloyal Customer    10000 non-null  uint8  
 22  Type of Travel_Personal Travel     10000 non-null  uint8  
 23  Class_Eco                          10000 non-null  uint8  
 24  Class_Eco Plus                     10000 non-null  uint8  
dtypes: float64(1), int64(19), uint8(5)
memory usage: 1.6 MB
```

In [10]:
```
# Step 2: Define our X and Y
Y = ASF[['satisfaction']]

X = ASF.drop(columns=['satisfaction','id'])
```

In [11]:
```
# Step 3: to split the data into train and test

from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, train_size=0.8, random_sta
```

In [12]:
```
len(X_train), len(X_test), len(Y_train), len(Y_test)
```

Out[12]:
```
(8000, 2000, 8000, 2000)
```

In [13]:
```
# Step 4: Building the model

# Create our model object
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=500)

# fit the object on the training the data (fit command)

model = rf.fit(X_train, Y_train)
model
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_44672\1582908817.py:10: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().
  model = rf.fit(X_train, Y_train)
```

Out[13]:
```
RandomForestClassifier(n_estimators=500)
```

In [14]:
```
# Step 5: Predicting the test cases
Y_test
Y_test['pred_RFC'] = model.predict(X_test)
Y_test
```

Out[14]:

	satisfaction	pred_RFC
2374	1	1
1784	0	0
6301	1	0
1600	0	0
7920	0	0
...	...	...
8623	0	0
5928	0	0
6714	1	1
5885	0	0
7289	0	0

2000 rows × 2 columns

In [15]:
```
# Step 6: We will build our confusion matrix to check the accuracy of the model

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

accuracy_score(Y_test['satisfaction'], Y_test['pred_RFC'])
```

Out[15]:
```
0.9465
```

In [16]:
```
confusion_matrix(Y_test['satisfaction'], Y_test['pred_RFC'])
```

Out[16]:
```
array([[1090,   42],
       [  65,  803]], dtype=int64)
```

In [17]:
```
print(classification_report(Y_test['satisfaction'], Y_test['pred_RFC']))
```

```
              precision    recall  f1-score   support

           0       0.94      0.96      0.95      1132
           1       0.95      0.93      0.94       868

    accuracy                           0.95      2000
   macro avg       0.95      0.94      0.95      2000
weighted avg       0.95      0.95      0.95      2000
```

Ans 1b

In [18]:
```
# we can reuse step 1 ) filling missing values, & step 2) defining X&Y from above
# after that, Since there are some columns where the maximum value and minimum valu
# we can standardize / scale our data using StandardScaler

# (X1 - average of column) / (maximum of column - minimum of column)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_scal = sc.fit_transform(X)
```

In [19]:
```
pd.DataFrame(X_scal)
```

Out[19]:

	0	1	2	3	4	5	6	7	
0	-1.740155	-0.737552	0.202159	0.628992	0.175602	-1.533190	1.337013	-0.177909	1.198
1	-0.946280	-0.962165	0.202159	-0.679908	0.175602	0.024218	-1.648724	-0.177909	-1.818
2	-0.880123	-0.056726	-0.550762	-0.679908	-0.537070	-0.754486	1.337013	1.302818	1.198
3	-0.946280	-0.635728	-0.550762	1.283442	1.600946	1.581625	-0.902290	-0.918273	-1.064
4	1.435347	-0.983129	0.202159	-0.025458	0.175602	0.024218	0.590579	1.302818	1.198
...	...	...	...	...	...	...	...	...	
9995	0.707627	2.396045	0.202159	-0.025458	0.175602	0.024218	0.590579	1.302818	0.443
9996	-0.086248	2.669574	1.708003	1.283442	1.600946	1.581625	1.337013	1.302818	1.198
9997	-0.020092	-0.878310	0.955081	0.628992	0.888274	0.802921	1.337013	0.562454	0.443
9998	0.839940	0.163894	1.708003	1.283442	1.600946	1.581625	-0.902290	1.302818	1.198
9999	0.112221	2.734462	0.955081	0.628992	0.888274	0.802921	-0.902290	0.562454	1.198

10000 rows × 23 columns

In [20]:

```
X
```

Out[20]:

	Age	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	Food and drink	Online boarding	Seat comfort	en
0	13	460	3	4	3	1	5	3	5	
1	25	235	3	2	3	3	1	3	1	
2	26	1142	2	2	2	2	5	5	5	
3	25	562	2	5	5	5	2	2	2	
4	61	214	3	3	3	3	4	5	5	
...	...	...	...	...	...	...	...	...	...	
9995	50	3599	3	3	3	3	4	5	4	
9996	38	3873	5	5	5	5	5	5	5	
9997	39	319	4	4	4	4	5	4	4	
9998	52	1363	5	5	5	5	2	5	5	
9999	41	3938	4	4	4	4	2	4	5	

10000 rows × 23 columns

In [21]:

```
# Step 3: split into train & test set
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X_scal, Y, train_size=0.8, ranc
```

In [22]:

```
# Step 4: building the model:
```

```
# create model object
from sklearn.svm import SVC
svm = SVC()

# fit the object on training data
model = svm.fit(X_train,Y_train)
model
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[22]: SVC()

In [23]:
```
# Predict the test cases

Y_test['pred_svm'] = model.predict(X_test)
```

In [24]:
```
Y_test
```

Out[24]:

	satisfaction	pred_svm
7830	0	0
5466	1	1
647	0	0
7801	1	1
8846	0	0
...	...	...
1287	0	0
768	0	0
6616	0	0
8385	0	0
8639	1	1

2000 rows × 2 columns

In [25]:
```
# Step 6: We will build our confusion matrix to check the accuracy of the model

#from sklearn.metrics import accuracy_score, confusion_matrix, classification_repor

accuracy_score(Y_test['satisfaction'], Y_test['pred_svm'])
```

Out[25]: 0.9405

In [26]:
```
confusion_matrix(Y_test['satisfaction'], Y_test['pred_svm'])
```

Out[26]:
```
array([[1119,   50],
       [  69,  762]], dtype=int64)
```

In [27]:
```
# Check the accuracy of the model
from sklearn.metrics import classification_report

print(classification_report(Y_test['satisfaction'], Y_test['pred_svm']))
```

```
              precision    recall  f1-score   support

           0       0.94      0.96      0.95      1169
           1       0.94      0.92      0.93       831

    accuracy                           0.94      2000
   macro avg       0.94      0.94      0.94      2000
weighted avg       0.94      0.94      0.94      2000
```

Conclusion

The results of RF & SVM are very similar, both models perform equally well. The data is not imbalanced, therefore the correct prediction of satisfaction - true or false are both equally probable

HR Data

Ans 2c

In [28]:
```
#import the data set
import pandas as pd
HR=pd.read_csv("hr_data.csv")
HR
```

Out[28]:

	employee_id	number_project	average_montly_hours	time_spend_company	Work_accident
0	1003	2	157	3	0
1	1005	5	262	6	0
2	1486	7	272	4	0
3	1038	5	223	5	0
4	1057	2	159	3	0
...	...	...	...	...	...
14994	87670	2	151	3	0
14995	87673	2	160	3	0
14996	87679	2	143	3	0
14997	87681	6	280	4	0
14998	87684	2	158	3	0

14999 rows × 9 columns

In [29]:
```
HR.drop(columns=['employee_id'], inplace=True)
```

In [30]:
```
HR.isnull().sum() # No null values
```

Out[30]:
```
number_project          0
average_montly_hours    0
time_spend_company      0
Work_accident           0
attrition               0
promotion_last_5years   0
department              0
salary                  0
dtype: int64
```

In [31]:
```
HR.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 8 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   number_project         14999 non-null  int64
 1   average_montly_hours   14999 non-null  int64
 2   time_spend_company     14999 non-null  int64
 3   Work_accident          14999 non-null  int64
 4   attrition              14999 non-null  object
 5   promotion_last_5years  14999 non-null  int64
 6   department             14999 non-null  object
 7   salary                 14999 non-null  object
dtypes: int64(5), object(3)
memory usage: 937.6+ KB
```

In [32]:
```
# we have 3 categorical variables, with no order of importance
# Applying Dummy variable (get_dummies) to convert the text columns into numerical

HRF= pd.get_dummies(HR, columns = ['attrition','department','salary'], drop_first =
HRF
```

Out[32]:

	number_project	average_montly_hours	time_spend_company	Work_accident	promotion_la
0	2	157	3	0	
1	5	262	6	0	
2	7	272	4	0	
3	5	223	5	0	
4	2	159	3	0	
...	...	...	...	...	
14994	2	151	3	0	
14995	2	160	3	0	
14996	2	143	3	0	
14997	6	280	4	0	
14998	2	158	3	0	

14999 rows × 17 columns

In [33]:
```
# 'attrition' was changed to 'attrition_stayed' because of get_dummies # renaming t
HRF.rename(columns={'attrition_stayed': 'attrition'}, inplace=True)
```

```
HRF
```

Out[33]:

	number_project	average_montly_hours	time_spend_company	Work_accident	promotion_la
0	2	157	3	0	
1	5	262	6	0	
2	7	272	4	0	
3	5	223	5	0	
4	2	159	3	0	
...	...	...	...	...	
14994	2	151	3	0	
14995	2	160	3	0	
14996	2	143	3	0	
14997	6	280	4	0	
14998	2	158	3	0	

14999 rows × 17 columns

In [34]:
```
# define X & Y # The target variable 'attrition' has changed to 'attrition_stayed'
# but that really does not make a difference because
Y= HRF[['attrition']]
X = HRF.drop(columns=['attrition'])
```

In [35]:
```
 #Since there are some columns where the maximum value and minimum value is in very
# we can standardize / scale our data using StandardScaler

# (X1 - average of column) / (maximum of column - minimum of column)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_scal = sc.fit_transform(X)
```

In [36]:
```
pd.DataFrame(X_scal)
```

Out[36]:

	0	1	2	3	4	5	6	7	
0	-1.462863	-0.882040	-0.341235	-0.411165	-0.147412	-0.235321	-0.232148	-0.227647	-0.20
1	0.971113	1.220423	1.713436	-0.411165	-0.147412	-0.235321	-0.232148	-0.227647	-0.20
2	2.593763	1.420657	0.343655	-0.411165	-0.147412	-0.235321	-0.232148	-0.227647	-0.20
3	0.971113	0.439508	1.028546	-0.411165	-0.147412	-0.235321	-0.232148	-0.227647	-0.20
4	-1.462863	-0.841993	-0.341235	-0.411165	-0.147412	-0.235321	-0.232148	-0.227647	-0.20
...	...	...	...	...	...	...	...	...	
14994	-1.462863	-1.002181	-0.341235	-0.411165	-0.147412	-0.235321	-0.232148	-0.227647	-0.20
14995	-1.462863	-0.821970	-0.341235	-0.411165	-0.147412	-0.235321	-0.232148	-0.227647	-0.20
14996	-1.462863	-1.162368	-0.341235	-0.411165	-0.147412	-0.235321	-0.232148	-0.227647	-0.20
14997	1.782438	1.580845	0.343655	-0.411165	-0.147412	-0.235321	-0.232148	-0.227647	-0.20
14998	-1.462863	-0.862016	-0.341235	-0.411165	-0.147412	-0.235321	-0.232148	-0.227647	-0.20

14999 rows × 16 columns

In [37]:
```
X
```

Out[37]:

	number_project	average_montly_hours	time_spend_company	Work_accident	promotion_las
0	2	157	3	0	
1	5	262	6	0	
2	7	272	4	0	
3	5	223	5	0	
4	2	159	3	0	
...	...	...	...	...	
14994	2	151	3	0	
14995	2	160	3	0	
14996	2	143	3	0	
14997	6	280	4	0	
14998	2	158	3	0	

14999 rows × 16 columns

In [38]:
```
# split into train & test set
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X_scal, Y, train_size=0.8, ranc
```

In [39]:
```
#building the model:

# create model object
from sklearn.svm import SVC
```

```
svm = SVC()

# fit the object on training data
model = svm.fit(X_train,Y_train)
model
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[39]:
```
SVC()
```

In [40]:
```
# Predict the test cases

Y_test['pred_svm'] = model.predict(X_test)
```

In [41]:
```
Y_test
```

Out[41]:

	attrition	pred_svm
7914	1	1
14558	0	0
12046	0	0
11354	1	1
12729	0	0
...	...	...
8933	1	1
3699	1	1
12477	0	0
162	0	0
12719	0	0

3000 rows × 2 columns

In [42]:
```
# Step 6: We will build our confusion matrix to check the accuracy of the model

#from sklearn.metrics import accuracy_score, confusion_matrix, classification_repor

accuracy_score(Y_test['attrition'], Y_test['pred_svm'])
```

Out[42]:
```
0.922
```

In [43]:
```
confusion_matrix(Y_test['attrition'], Y_test['pred_svm'])
```

Out[43]:
```
array([[ 615,  120],
       [ 114, 2151]], dtype=int64)
```

In [44]:
```
# Check the accuracy of the model
from sklearn.metrics import classification_report

print(classification_report(Y_test['attrition'], Y_test['pred_svm']))
```

```
              precision    recall  f1-score   support

           0       0.84      0.84      0.84       735
           1       0.95      0.95      0.95      2265

    accuracy                           0.92      3000
   macro avg       0.90      0.89      0.89      3000
weighted avg       0.92      0.92      0.92      3000
```

Ans 2d

In [45]:
```
HRF['attrition'].value_counts()
```

Out[45]:
```
1    11428
0     3571
Name: attrition, dtype: int64
```

In [46]:
```
# applying normal LogisticRegression
```

In [47]:
```
# Create our model object
from sklearn.linear_model import LogisticRegression

LR = LogisticRegression()

# fit the object on training data

model = LR.fit(X_train, Y_train)
model
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[47]:
```
LogisticRegression()
```

In [48]:
```
# predicting the valye
Y_test['pred_LR'] = model.predict(X_test)
```

In [49]:
```
Y_test
```

Out[49]:

	attrition	pred_svm	pred_LR
7914	1	1	1
14558	0	0	1
12046	0	0	1
11354	1	1	1
12729	0	0	1
...	...	...	...
8933	1	1	1
3699	1	1	1
12477	0	0	1
162	0	0	1
12719	0	0	0

3000 rows × 3 columns

In [50]:
```
accuracy_score(Y_test['attrition'], Y_test['pred_LR'])
```

Out[50]:
```
0.743
```

In [51]:
```
confusion_matrix(Y_test['attrition'], Y_test['pred_LR'])
```

Out[51]:
```
array([[  24,  711],
       [  60, 2205]], dtype=int64)
```

In [52]:
```
# Check the accuracy of the model

print(classification_report(Y_test['attrition'], Y_test['pred_LR']))
```

```
              precision    recall  f1-score   support

           0       0.29      0.03      0.06       735
           1       0.76      0.97      0.85      2265

    accuracy                           0.74      3000
   macro avg       0.52      0.50      0.45      3000
weighted avg       0.64      0.74      0.66      3000
```

from the results it is observed that the precision, recall & f-1 score of attrition =0 is far lower than attrition =1 this is because the data is imbalanced there are far more occurances of attrition =1 (i.e. people left the org) than attrition=0 (people stayed)! LogisticRegression does not perform well in these conditions Therefore SMOT must be used

LogisticRegression with SMOT

In [53]:
```
# lets find the number of datapoints in training dataset for target variable
Y_train['attrition'].value_counts()
# we can see that the data is imbalanced
```

Out[53]:
```
1    9163
0    2836
Name: attrition, dtype: int64
```

In [54]:
```
# import the functionality of smote

from imblearn.over_sampling import SMOTE

# we will balance the training data X_train and Y_train

# we will create a smote object

sm = SMOTE(random_state = 9999)

# we will fit the smote object on X_train and Y_train ---> fit.resample

X_train_new, Y_train_new = sm.fit_resample(X_train, Y_train)
```

In [55]:
```
# lets again find the number of datapoints in the balanced training dataset for tar
Y_train_new['attrition'].value_counts()
# we can see that the data is now balanced
```

Out[55]:
```
0    9163
1    9163
Name: attrition, dtype: int64
```

In [56]:
```
# Step 4: Create the model using training dataset

from sklearn.linear_model import LogisticRegression

LR = LogisticRegression()

model = LR.fit(X_train_new,Y_train_new)

model
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[56]:
```
LogisticRegression()
```

In [57]:
```
#  Predict the Test cases

Y_test['pred_LRS'] = model.predict(X_test)
```

In [58]:
```
Y_test
```

Out[58]:

	attrition	pred_svm	pred_LR	pred_LRS
7914	1	1	1	0
14558	0	0	1	0
12046	0	0	1	0
11354	1	1	1	0
12729	0	0	1	0
...	...	...	...	...
8933	1	1	1	1
3699	1	1	1	1
12477	0	0	1	0
162	0	0	1	0
12719	0	0	0	0

3000 rows × 4 columns

In [59]:
```
accuracy_score(Y_test['attrition'], Y_test['pred_LRS'])
```
Out[59]:
```
0.635
```

In [60]:
```
confusion_matrix(Y_test['attrition'], Y_test['pred_LRS'])
```
Out[60]:
```
array([[ 534,  201],
       [ 894, 1371]], dtype=int64)
```

In [61]:
```
# Check the accuracy of the model

print(classification_report(Y_test['attrition'], Y_test['pred_LRS']))
```
```
              precision    recall  f1-score   support

           0       0.37      0.73      0.49       735
           1       0.87      0.61      0.71      2265

    accuracy                           0.64      3000
   macro avg       0.62      0.67      0.60      3000
weighted avg       0.75      0.64      0.66      3000
```

Conclusion

LogisticRegression with SMOT performs better than LogisticRegression without SMOT (recall values are closer to each other)

but LogisticRegression as a model (with and without SMOT) performs inferior to SVM in this case

Bollywood

Ans 3a

In [62]:
```
# Step 1: Read and access data

import pandas as pd

bo = pd.read_csv('bollywood.csv')
bo
```

Out[62]:

	SlNo	Release Date	MovieName	ReleaseTime	Genre	Budget	BoxOfficeCollection	YoutubeVi
0	1	18-Apr-14	2 States	LW	Romance	36	104.00	8576
1	2	4-Jan-13	Table No. 21	N	Thriller	10	12.00	1087
2	3	18-Jul-14	Amit Sahni Ki List	N	Comedy	10	4.00	572
3	4	4-Jan-13	Rajdhani Express	N	Drama	7	0.35	42
4	5	4-Jul-14	Bobby Jasoos	N	Comedy	18	10.80	3113
...	...	...	...	...	...	...	...	
144	145	27-Feb-15	Dum Laga Ke Haisha	N	Comedy	15	30.00	3250
145	146	13-Mar-15	NH10	N	Thriller	13	32.10	5592
146	147	20-Mar-15	Dilliwali Zaalim Girlfriend	N	Comedy	32	12.00	2316
147	148	20-Mar-15	Hunterrr	N	Comedy	5	11.89	4674
148	149	23-May-14	Kochadaiiyaan	HS	Action	150	120.00	4740

149 rows × 10 columns

In [63]:
```
bo.isnull().sum()
```

Out[63]:
```
SlNo                 0
Release Date         0
MovieName            0
ReleaseTime          0
Genre                0
Budget               0
BoxOfficeCollection  0
YoutubeViews         0
YoutubeLikes         0
YoutubeDislikes      0
dtype: int64
```

In [64]:
```
bo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 149 entries, 0 to 148
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   SlNo               149 non-null    int64  
 1   Release Date       149 non-null    object 
 2   MovieName          149 non-null    object 
 3   ReleaseTime        149 non-null    object 
 4   Genre              149 non-null    object 
 5   Budget             149 non-null    int64  
 6   BoxOfficeCollection  149 non-null    float64
 7   YoutubeViews       149 non-null    int64  
 8   YoutubeLikes       149 non-null    int64  
 9   YoutubeDislikes    149 non-null    int64  
dtypes: float64(1), int64(5), object(4)
memory usage: 11.8+ KB
```

- assuming that SlNo, Release Date, MovieName have no effect on Box Office collections, that can be modelled with a Linear Regression, and can be ignored
- we have 2 categorical columns that could have an effect on Box Office Collections, viz : ReleaseTime & Genre, therefore we will convert them to numerical using get_dummies
- there are multiple input values to model one output --- This is a Multiple Linear Regression problem

In [65]:
```
bo1= pd.get_dummies(bo, columns = ['ReleaseTime','Genre'], drop_first = True)
```

In [66]:
```
bo1
```

Out[66]:

	SlNo	Release Date	MovieName	Budget	BoxOfficeCollection	YoutubeViews	YoutubeLikes	You
0	1	18-Apr-14	2 States	36	104.00	8576361	26622	
1	2	4-Jan-13	Table No. 21	10	12.00	1087320	1129	
2	3	18-Jul-14	Amit Sahni Ki List	10	4.00	572336	586	
3	4	4-Jan-13	Rajdhani Express	7	0.35	42626	86	
4	5	4-Jul-14	Bobby Jasoos	18	10.80	3113427	4512	
...	...	...	...	...	...	...	...	
144	145	27-Feb-15	Dum Laga Ke Haisha	15	30.00	3250917	8185	
145	146	13-Mar-15	NH10	13	32.10	5592977	15464	
146	147	20-Mar-15	Dilliwali Zaalim Girlfriend	32	12.00	2316047	4289	
147	148	20-Mar-15	Hunterrr	5	11.89	4674795	3706	
148	149	23-May-14	Kochadaiiyaan	150	120.00	4740727	13466	

149 rows × 17 columns

In [67]:

```
#Spliting the data into X and Y
X = bo1.drop(['SlNo', 'Release Date', 'MovieName', 'BoxOfficeCollection'], axis = 1
Y = bo1[['BoxOfficeCollection']]
```

In [68]:

```
Y
```

Out[68]:

	BoxOfficeCollection
0	104.00
1	12.00
2	4.00
3	0.35
4	10.80
...	...
144	30.00
145	32.10
146	12.00
147	11.89
148	120.00

149 rows × 1 columns

In [69]:
```
#Dividing the data into training and test data
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_sta
len(X_train), len(X_test), len(Y_train), len(Y_test)
```

Out[69]:
```
(119, 30, 119, 30)
```

In [70]:
```
#Creating the model using on the training data set
from sklearn.linear_model import LinearRegression
lr = LinearRegression()

#Fit the model object into training data to build a model
model=lr.fit(X_train, Y_train)
model
```

Out[70]:
```
LinearRegression()
```

Find the values of slope, intercept and R Square (Applicable only when we build Regression based model) For linear model R-sq >=75

In [71]:
```
# To calculate the slope
model.coef_
```

Out[71]:
```
array([[ 1.67045158e+00, -2.08110826e-06,  4.77579135e-03,
        -7.90521961e-03,  3.05203376e+01,  1.74763032e+01,
        -1.04729815e+00, -6.81932554e+01, -4.12776923e+01,
        -1.08308774e+01, -1.40448963e+01, -2.40587144e+01,
        -1.24776424e+01]])
```

In [72]:
```
# To calculate constant
model.intercept_
```

Out[72]:
```
array([1.18908213])
```

In [73]:
```
# To calculate the R-sq value for model
rsq = model.score(X_train, Y_train)
```

```
rsq
```

Out[73]: 0.5949046896114394

In [74]:
```
Y_test['predicted'] = model.predict(X_test)
Y_test
```

Out[74]:

	BoxOfficeCollection	predicted
111	10.00	-1.458531
75	37.80	94.781344
136	56.00	26.020935
56	36.00	76.717975
110	35.50	48.715068
9	0.01	3.470497
65	3.75	33.393999
15	0.09	-61.501217
30	1.50	11.980225
63	0.70	7.627056
88	14.05	10.127115
62	45.00	24.292409
83	40.00	46.540339
112	4.00	14.156968
138	125.00	136.087759
41	111.00	140.692088
102	4.00	55.714393
66	2.50	7.512839
90	100.00	84.412100
141	38.00	83.965879
69	61.00	57.694435
2	4.00	7.196061
22	76.70	36.997091
51	8.78	-9.962917
12	105.50	122.828194
11	10.25	-4.411023
140	14.02	2.101446
133	22.00	57.151939
16	162.00	133.867196
84	55.00	44.303060

In [75]:
```
# Step 7: Calculate the RMSE value from test data. RMSE - Root <- Mean <- Square <-
Y_test['error'] = Y_test['BoxOfficeCollection'] - Y_test['predicted']
```

```
Y_test['sq-error'] = (Y_test['BoxOfficeCollection'] - Y_test['predicted']) **2

Error_mean = Y_test['sq-error'].mean()
Error_mean #This is mean of sq_Error

#Find the root of Error mean -> RMSE
import math
RMSE = math.sqrt(Error_mean)
RMSE
```

Out[75]:

```
27.294169764305966
```

we know from theory that A Linear Regression model with HIGH value of R Square and LOW RMSE is an ideal model. however we observe here that the Rsquare = 60(approx) which is not very high and RMSE = 27 (approx) which is not considered very low :(

We can therefore conclude that the Box Office performance of a movie cannot be very well explained/ predicted by a LinearRegression model

In [76]:

```
Y_test
```

Out[76]:

	BoxOfficeCollection	predicted	error	sq-error
111	10.00	-1.458531	11.458531	131.297934
75	37.80	94.781344	-56.981344	3246.873544
136	56.00	26.020935	29.979065	898.744360
56	36.00	76.717975	-40.717975	1657.953462
110	35.50	48.715068	-13.215068	174.638027
9	0.01	3.470497	-3.460497	11.975036
65	3.75	33.393999	-29.643999	878.766705
15	0.09	-61.501217	61.591217	3793.477965
30	1.50	11.980225	-10.480225	109.835108
63	0.70	7.627056	-6.927056	47.984112
88	14.05	10.127115	3.922885	15.389027
62	45.00	24.292409	20.707591	428.804329
83	40.00	46.540339	-6.540339	42.776030
112	4.00	14.156968	-10.156968	103.163999
138	125.00	136.087759	-11.087759	122.938400
41	111.00	140.692088	-29.692088	881.620095
102	4.00	55.714393	-51.714393	2674.378495
66	2.50	7.512839	-5.012839	25.128554
90	100.00	84.412100	15.587900	242.982630
141	38.00	83.965879	-45.965879	2112.862037
69	61.00	57.694435	3.305565	10.926763
2	4.00	7.196061	-3.196061	10.214806
22	76.70	36.997091	39.702909	1576.320945
51	8.78	-9.962917	18.742917	351.296951
12	105.50	122.828194	-17.328194	300.266297
11	10.25	-4.411023	14.661023	214.945592
140	14.02	2.101446	11.918554	142.051921
133	22.00	57.151939	-35.151939	1235.658787
16	162.00	133.867196	28.132804	791.454650
84	55.00	44.303060	10.696940	114.424534

In [ ]: