In [1]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
df = pd.read_csv('Social Media Data for DSBA.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	UserID	Taken_product	Yearly_avg_view_on_travel_page	preferred_device	total_likes_on_outstation_checkin_given	yearly_a
0	1000001	Yes	307.0	iOS and Android	38570.0	
1	1000002	No	367.0	iOS	9765.0	
2	1000003	Yes	277.0	iOS and Android	48055.0	
3	1000004	No	247.0	iOS	48720.0	
4	1000005	No	202.0	iOS and Android	20685.0	

In [4]:

```
df.tail()
```

Out[4]:

	UserID	Taken_product	Yearly_avg_view_on_travel_page	preferred_device	total_likes_on_outstation_checkin_given	yea
11755	1011756	No	279.0	Laptop	30987.0	
11756	1011757	No	305.0	Tab	21510.0	
11757	1011758	No	214.0	Tab	5478.0	
11758	1011759	No	382.0	Laptop	35851.0	
11759	1011760	No	270.0	Tab	22025.0	

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11760 entries, 0 to 11759
Data columns (total 17 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   UserID                                  11760 non-null  int64
 1   Taken_product                           11760 non-null  object
 2   Yearly_avg_view_on_travel_page          11179 non-null  float64
 3   preferred_device                        11707 non-null  object
 4   total_likes_on_outstation_checkin_given 11379 non-null  float64
 5   yearly avg Outstation checkins          11685 non-null  object
```

```
 6   member_in_family                          11760 non-null  object
 7   preferred_location_type                   11729 non-null  object
 8   Yearly_avg_comment_on_travel_page         11554 non-null  float64
 9   total_likes_on_outofstation_checkin_received  11760 non-null  int64
 10  week_since_last_outstation_checkin        11760 non-null  int64
 11  following_company_page                    11657 non-null  object
 12  montly_avg_comment_on_company_page        11760 non-null  int64
 13  working_flag                              11760 non-null  object
 14  travelling_network_rating                 11760 non-null  int64
 15  Adult_flag                                11760 non-null  int64
 16  Daily_Avg_mins_spend_on_traveling_page    11760 non-null  int64
dtypes: float64(3), int64(7), object(7)
memory usage: 1.5+ MB
```

In [6]:

```
df.describe(include='all')
```

Out[6]:

	UserID	Taken_product	Yearly_avg_view_on_travel_page	preferred_device	total_likes_on_outstation_checkin_give
count	1.176000e+04	11760	11179.000000	11707	11379.00000
unique	NaN	2	NaN	10	Na
top	NaN	No	NaN	Tab	Na
freq	NaN	9864	NaN	4172	Na
mean	1.005880e+06	NaN	280.830844	NaN	28170.48176
std	3.394964e+03	NaN	68.182958	NaN	14385.03213
min	1.000001e+06	NaN	35.000000	NaN	3570.00000
25%	1.002941e+06	NaN	232.000000	NaN	16380.00000
50%	1.005880e+06	NaN	271.000000	NaN	28076.00000
75%	1.008820e+06	NaN	324.000000	NaN	40525.00000
max	1.011760e+06	NaN	464.000000	NaN	252430.00000

In [7]:

```
print('The number of rows in the data are',df.shape[0],'\n''The number of columns in the data are',df.shape[1])
```

```
The number of rows in the data are 11760
The number of columns in the data are 17
```

In [8]:

```
print(df.isnull().sum())
print('\n')
print('The total number of null values in the data are',df.isnull().sum().sum())
```

```
UserID                                        0
Taken_product                                 0
Yearly_avg_view_on_travel_page              581
preferred_device                             53
total_likes_on_outstation_checkin_given     381
yearly_avg_Outstation_checkins               75
member_in_family                              0
preferred_location_type                      31
Yearly_avg_comment_on_travel_page           206
total_likes_on_outofstation_checkin_received  0
week_since_last_outstation_checkin            0
following_company_page                      103
montly_avg_comment_on_company_page            0
working_flag                                  0
travelling_network_rating                     0
Adult_flag                                    0
```

```
Daily_Avg_Mins_Spend_On_Travelling_Page            0
dtype: int64


The total number of null values in the data are 1430
```

In [9]:

```
print('The total number of duplicate values are',df.duplicated().sum())
```

```
The total number of duplicate values are 0
```

In [10]:

```
for col in df.columns:
    print('\n')
    print('The name of the column is',col)
    print(df[col].value_counts(sort='ascending'))
```

```
The name of the column is UserID
1001471    1
1002875    1
1006969    1
1004920    1
1011063    1
          ..
1000165    1
1002212    1
1008353    1
1010400    1
1001473    1
Name: UserID, Length: 11760, dtype: int64


The name of the column is Taken_product
No     9864
Yes    1896
Name: Taken_product, dtype: int64


The name of the column is Yearly_avg_view_on_travel_page
262.0    190
255.0    186
270.0    179
217.0    165
232.0    160
        ...
462.0      2
464.0      1
146.0      1
463.0      1
458.0      1
Name: Yearly_avg_view_on_travel_page, Length: 331, dtype: int64


The name of the column is preferred_device
Tab                4172
iOS and Android    4134
Laptop             1108
iOS                1095
Mobile              600
Android             315
Android OS          145
ANDROID             134
Others                2
Other                 2
Name: preferred_device, dtype: int64


The name of the column is total_likes_on_outstation_checkin_given
24185.0    12
11515.0    11
```

```
11515.0    11
18550.0    10
37870.0    10
34195.0     9
           ..
44172.0     1
47684.0     1
3783.0      1
15732.0     1
50795.0     1
Name: total_likes_on_outstation_checkin_given, Length: 7888, dtype: int64


The name of the column is yearly_avg_Outstation_checkins
1     4543
2      844
10     682
9      340
3      336
7      336
8      320
5      261
4      256
16     255
6      236
11     229
24     223
29     215
23     215
18     208
15     206
20     199
26     199
25     198
28     180
19     176
14     167
17     160
12     159
22     152
13     150
21     143
27      96
*        1
Name: yearly_avg_Outstation_checkins, dtype: int64


The name of the column is member_in_family
3        4561
4        3184
2        2256
1        1349
5         384
Three      15
10         11
Name: member_in_family, dtype: int64


The name of the column is preferred_location_type
Beach              2424
Financial          2409
Historical site    1856
Medical            1845
Other               643
Big Cities          636
Social media        633
Trekking            528
Entertainment       516
Hill Stations       108
Tour  Travel         60
Tour and Travel      47
Game                 12
OTT                   7
```

```
Off                   7
Movie                 5
Name: preferred_location_type, dtype: int64


The name of the column is Yearly_avg_comment_on_travel_page
96.0     192
66.0     191
90.0     190
56.0     188
80.0     184
        ...
124.0      3
685.0      1
215.0      1
615.0      1
815.0      1
Name: Yearly_avg_comment_on_travel_page, Length: 100, dtype: int64


The name of the column is total_likes_on_outofstation_checkin_received
2377     12
2380     11
2342     11
2096     10
2610     10
         ..
19245     1
9000      1
4902      1
11033     1
4026      1
Name: total_likes_on_outofstation_checkin_received, Length: 6288, dtype: int64


The name of the column is week_since_last_outstation_checkin
1     3070
3     1766
2     1700
4     1118
0     1032
5      728
6      654
7      594
9      472
8      428
10     138
11      60
Name: week_since_last_outstation_checkin, dtype: int64


The name of the column is following_company_page
No     8355
Yes    3285
1        12
0         5
Name: following_company_page, dtype: int64


The name of the column is montly_avg_comment_on_company_page
23     673
22     653
25     609
24     605
21     594
      ...
420      1
428      1
468      1
500      1
499      1
Name: montly_avg_comment_on_company_page, Length: 160, dtype: int64
```

```
The name of the column is working_flag
No     9952
Yes    1808
Name: working_flag, dtype: int64


The name of the column is travelling_network_rating
3    3672
4    3456
2    2424
1    2208
Name: travelling_network_rating, dtype: int64


The name of the column is Adult_flag
0    5048
1    4768
2    1264
3     680
Name: Adult_flag, dtype: int64


The name of the column is Daily_Avg_mins_spend_on_traveling_page
10     1126
9       676
8       662
6       624
7       554
13      532
11      530
12      500
14      496
15      480
5       444
16      444
17      430
1       336
4       330
18      322
20      292
19      288
21      258
22      254
23      238
3       218
24      184
25      150
2       146
28      142
26      140
29      134
27      116
32      102
31       82
30       74
34       62
33       60
36       56
35       48
0        46
37       46
40       32
38       30
39       26
41       20
44        8
42        6
43        4
45        4
46        3
125       1
```

```
135       1
170       1
47        1
270       1
235       1
Name: Daily_Avg_mins_spend_on_traveling_page, dtype: int64
```

In [11]:

```
for col in df.columns:
    print('\n')
    print('The name of the column is',col)
    print(df[col].unique())
    print(df[col].dtype)
```

```
The name of the column is UserID
[1000001 1000002 1000003 ... 1011758 1011759 1011760]
int64


The name of the column is Taken_product
['Yes' 'No']
object


The name of the column is Yearly_avg_view_on_travel_page
[307. 367. 277. 247. 202. 240.  nan 225. 285. 270. 262. 217. 232. 255.
 210. 165. 397. 180. 157. 330. 345. 292. 322. 375. 195. 360. 412. 382.
 300. 405. 435. 150. 187.  42. 427. 352.  35. 450. 135. 308. 368. 249.
 205. 445. 226. 287. 271. 263. 219. 234. 256. 212. 241. 399. 286. 182.
 159. 316. 332. 347. 248. 331. 294. 323. 376. 265. 204. 309. 257. 346.
 264. 361. 196. 278. 444. 272. 414. 339. 443. 233. 280. 422. 174. 384.
 302. 242. 181. 211. 436. 279. 151. 377. 188. 189. 166. 406. 324. 197.
 143. 167. 383. 227. 144. 301. 429. 203. 250. 413. 338. 310. 392. 317.
 354. 400. 369. 137. 136. 295. 391. 353. 218. 362. 428. 451. 430. 173.
 190. 398. 355. 437. 407. 152. 421. 158. 293. 235. 220. 340. 385. 370.
 325. 415. 452. 315. 379. 290. 231. 281. 269. 222. 244. 221. 246. 402.
 184. 276. 258. 168. 259. 404. 318. 333. 252. 304. 378. 273. 282. 253.
 199. 215. 228. 356. 268. 366. 335. 266. 254. 274. 291. 448. 417. 455.
 229. 236. 275. 349. 418. 425. 185. 389. 388. 411. 216. 446. 337. 283.
 243. 154. 261. 289. 193. 175. 409. 326. 386. 207. 153. 172. 372. 390.
 245. 313. 306. 230. 441. 176. 213. 433. 208. 387. 147. 342. 424. 148.
 408. 303. 395. 209. 319. 373. 267. 296. 350. 341. 297. 239. 401. 454.
 251. 299. 312. 344. 348. 305. 394. 238. 200. 363. 206. 420. 201. 284.
 142. 374. 298. 357. 334. 179. 364. 321. 161. 431. 351. 381. 311. 393.
 288. 329. 328. 192. 237. 214. 198. 396. 456. 327. 223. 458. 260. 141.
 380. 191. 160. 224. 457. 359. 320. 423. 410. 194. 365. 177. 403. 438.
 164. 170. 149. 138. 453. 169. 146. 447. 155. 162. 449. 439. 416. 145.
 358. 371. 343. 140. 336. 183. 314. 426. 419. 186. 440. 459. 156. 163.
 461. 178. 442. 432. 434. 462. 460. 171. 464. 463.]
float64


The name of the column is preferred_device
['iOS and Android' 'iOS' 'ANDROID' nan 'Android' 'Android OS' 'Other'
 'Others' 'Tab' 'Laptop' 'Mobile']
object


The name of the column is total_likes_on_outstation_checkin_given
[38570.  9765. 48055. ...  5478. 35851. 22025.]
float64


The name of the column is yearly_avg_Outstation_checkins
['1' '24' '23' '27' '16' '15' '26' '19' '21' '11' '10' '25' '12' '18' '29'
 nan '22' '14' '20' '28' '17' '13' '*' '5' '8' '2' '3' '9' '7' '6' '4']
object


The name of the column is member_in_family
['3' '1' '4' 'Three' '2' '5' '10']
```

['2' '1' '4' 'three' '3' '5' '10']
object

The name of the column is preferred_location_type
['Financial' 'Other' 'Medical' nan 'Game' 'Social media' 'Entertainment'
 'Tour and Travel' 'Movie' 'OTT' 'Tour Travel' 'Beach' 'Historical site'
 'Big Cities' 'Trekking' 'Hill Stations']
object

The name of the column is Yearly_avg_comment_on_travel_page
[94. 61. 92. 56. 40. 79. 81. 67. 44. 84. 49. 31. 93. 50.
  51. 80. 96. 78. 45. 82. 53. 83. 58. 72. 48. 42. 41. 86.
  97. 75. 33. 37. 73. nan 98. 47. 71.   3. 43. 99. 59. 95.
  57. 76. 87. 66. 55. 32. 52. 70. 62. 64. 63. 60. 100. 46.
  39. 77. 91. 54. 34. 90. 65. 36. 88. 35. 89. 68. 85. 69.
  74. 38. 106. 105. 103. 108. 111. 104. 102. 109. 110. 112. 101. 107.
 615. 114. 113. 215. 815. 685. 118. 117. 115. 116. 121. 122. 120. 124.
 119. 125. 123.]
float64

The name of the column is total_likes_on_outofstation_checkin_received
[5993 5130 2090 ... 12093 9983 6203]
int64

The name of the column is week_since_last_outstation_checkin
[8 1 6 9 0 4 5 2 7 3 10 11]
int64

The name of the column is following_company_page
['Yes' 'No' nan '1' '0']
object

The name of the column is montly_avg_comment_on_company_page
[11 23 15 12 13 20 22 21 17 14 16 18 19 24 25 30 29 28
  27 376 381 26 427 437 499 363 425 439 301 461 322 324 355 338 332 459
 460 453 300 474 368 352 445 310 323 490 371 444 343 417 393 463 350 432
 412 379 336 441 346 317 406 485 400 483 478 438 354 313 497 325 419 388
 398 378 397 349 356 420 347 500 442 435 447 484 330 326 360 403 465 365
 353 429 345 321 491 476 475 487 316 428 472 314 405 473 339 342 455 469
 399 422 370 361 467 458 304 410 383 466 446 302 486 333 418 351 391 468
 454 329 390 384 404 402 424 488 440 312 449 477 380 357 414 337 33 32
  31 34 35 36 37 40 38 41 39 43 42 45 44 47 46 48]
int64

The name of the column is working_flag
['No' 'Yes']
object

The name of the column is travelling_network_rating
[1 4 2 3]
int64

The name of the column is Adult_flag
[0 1 3 2]
int64

The name of the column is Daily_Avg_mins_spend_on_traveling_page
[8 10   7   6 12   1 17   5   3 31 13   0 26 24 22   9 19   2
  23 14 15   4 29 28 21 25 20 11 16 37 38 30 40 18 36 34
  32 33 35 27 41 135 45 43 39 44 42 170 235 270 47 46]
int64

Data Cleaning

In [12]:

```
df['yearly_avg_Outstation_checkins'] = np.where((df['yearly_avg_Outstation_checkins']=='
*'),df['yearly_avg_Outstation_checkins'].mode(),df['yearly_avg_Outstation_checkins'])
```

In [13]:

```
df['member_in_family'] = np.where(df['member_in_family']=='Three','3',df['member_in_fami
ly'])
```

In [14]:

```
# Considering No as 0 and Yes as 1 Then
df['following_company_page'] = np.where(df['following_company_page']=='1','Yes',df['foll
owing_company_page'])
df['following_company_page'] = np.where(df['following_company_page']=='0','No',df['follo
wing_company_page'])
```

In [15]:

```
df['preferred_device'] = np.where(df['preferred_device']=='ANDROID','Android',df['prefer
red_device'])
df['preferred_device'] = np.where(df['preferred_device']=='Other','Others',df['preferred
_device'])
```

In [16]:

```
df['preferred_location_type'] = np.where(df['preferred_location_type']=='Tour  Travel','
Tour and Travel',df['preferred_location_type'])
```

In [17]:

```
df1 = df.drop('UserID',axis=1)
```

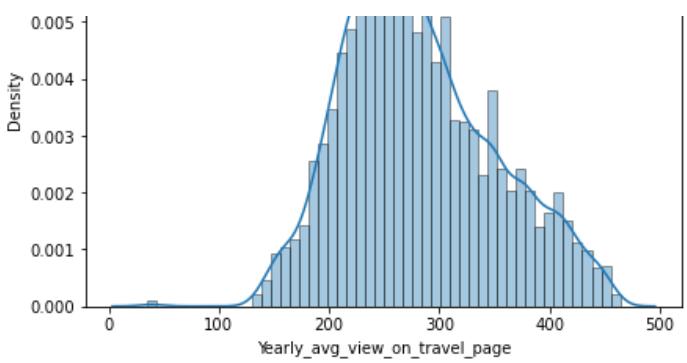
In [18]:

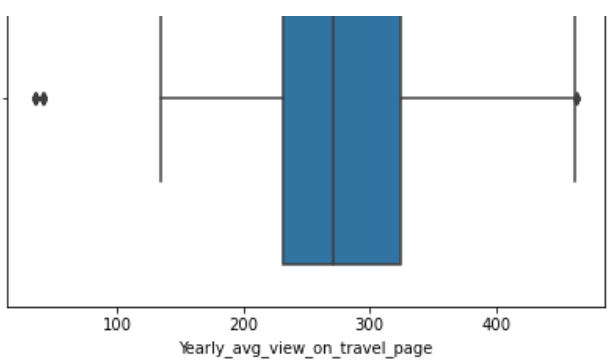
```
df1['Adult_flag'] = df1['Adult_flag'].astype('object')
df1['travelling_network_rating'] = df1['travelling_network_rating'].astype('category')
```

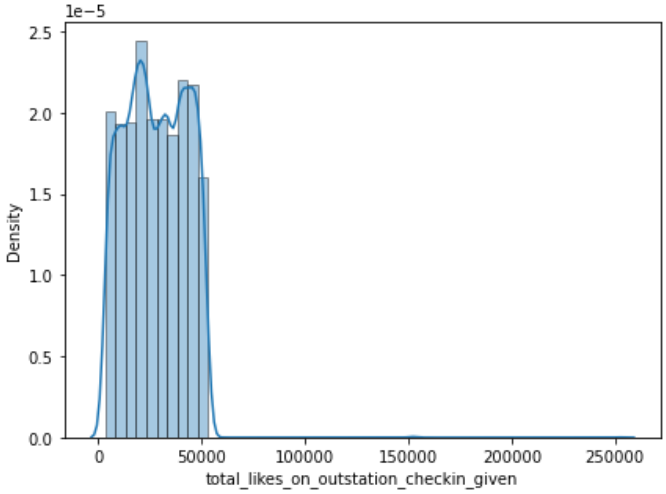
Univariate and Bivariate Analysis

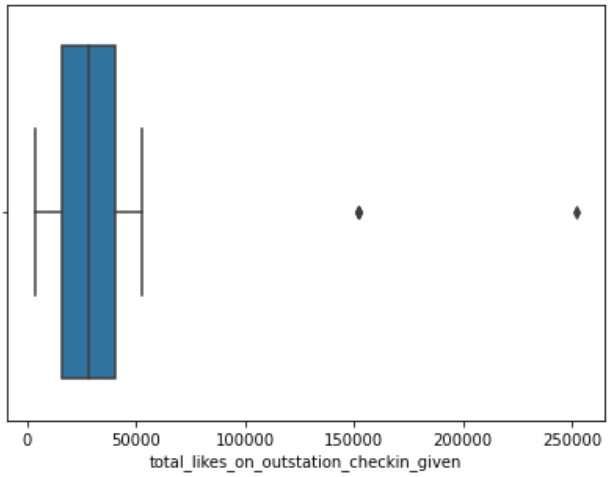
In [19]:

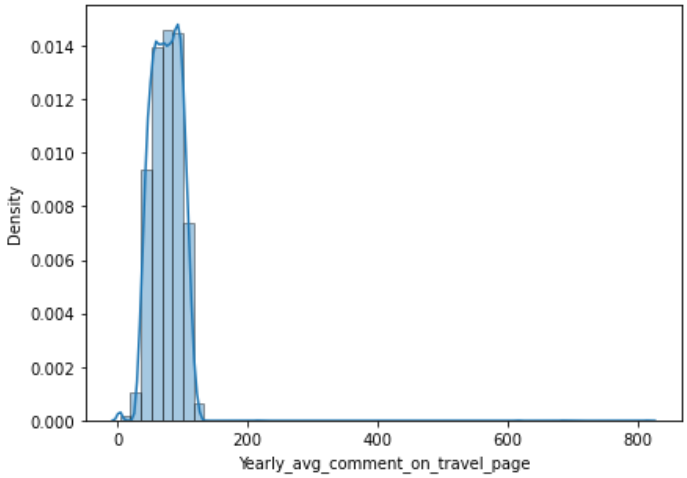
```
df_1 = df1.select_dtypes(['int64','float64'])

a = len(df_1.columns)     # number of rows
b = 2                     # number of columns
c = 1                    # initialize plot counter

fig = plt.figure(figsize=(15,40))

for i in df_1.columns:
    plt.subplot(a, b, c)
    plt.xlabel(i)
    sns.distplot(df_1[i], hist_kws= dict(ec = 'black'))
    c = c + 1

    plt.subplot(a, b, c)
    plt.xlabel(i)
    sns.boxplot(x = df_1[i])
    c = c + 1
```

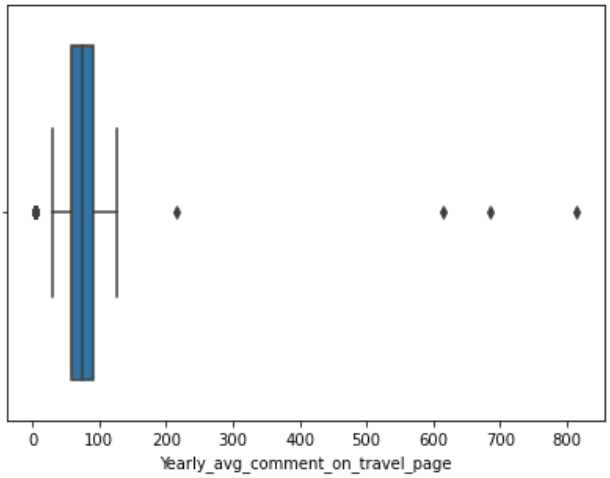

Density
Yearly_avg_view_on_travel_page

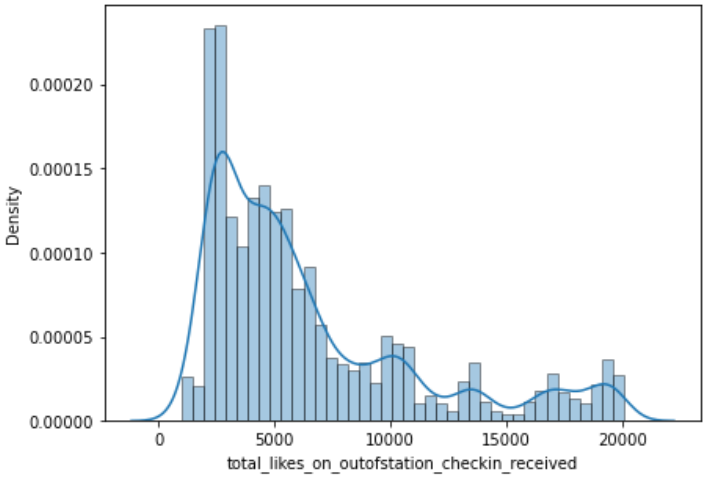

Yearly_avg_view_on_travel_page

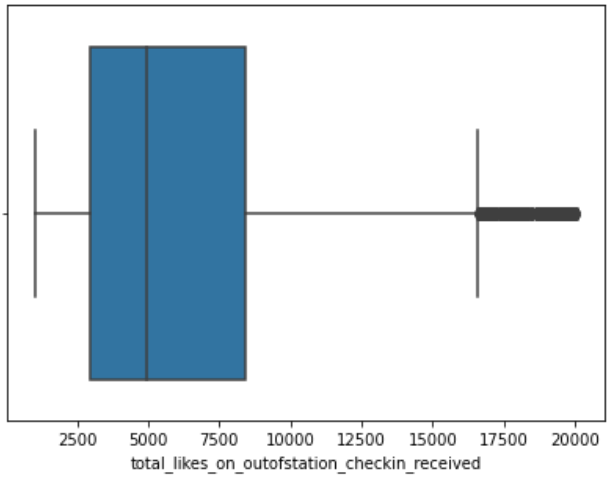

1e−5
Density
total_likes_on_outstation_checkin_given

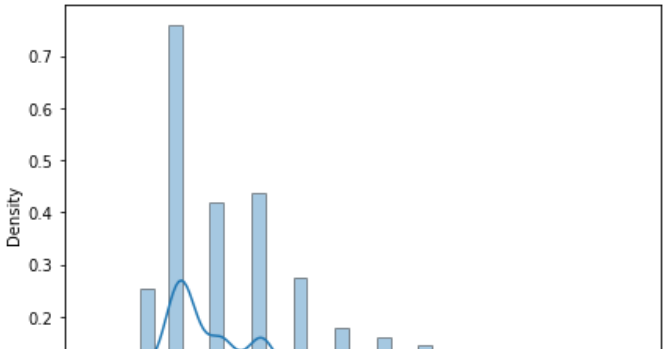

total_likes_on_outstation_checkin_given

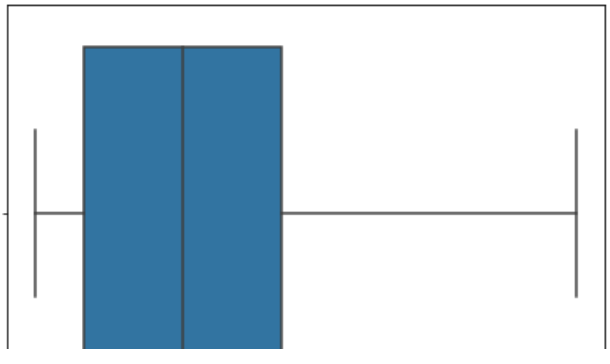

Density
Yearly_avg_comment_on_travel_page

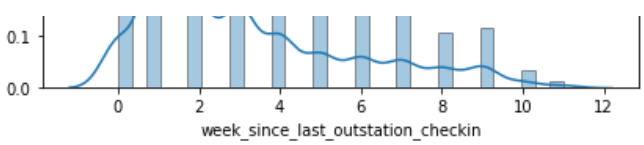

Yearly_avg_comment_on_travel_page

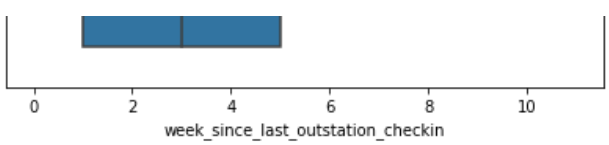

Density
total_likes_on_outofstation_checkin_received

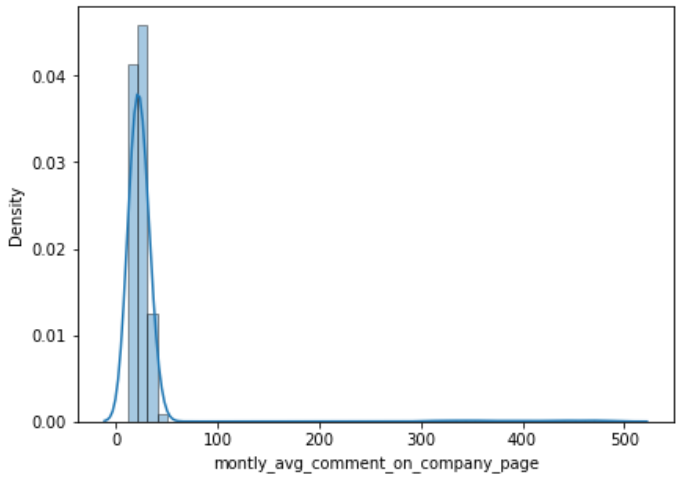

total_likes_on_outofstation_checkin_received

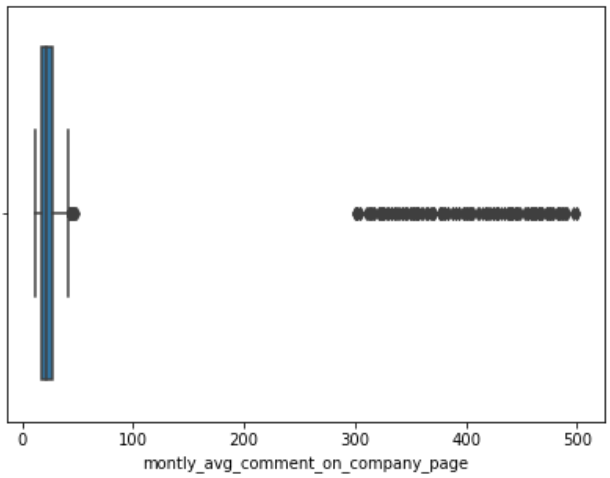

Density

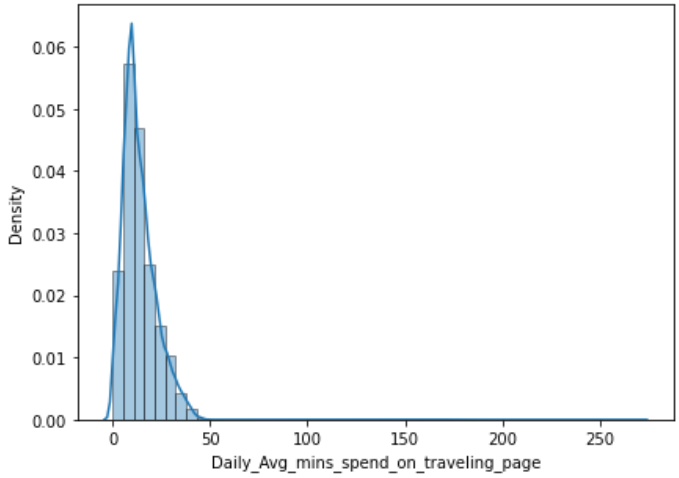


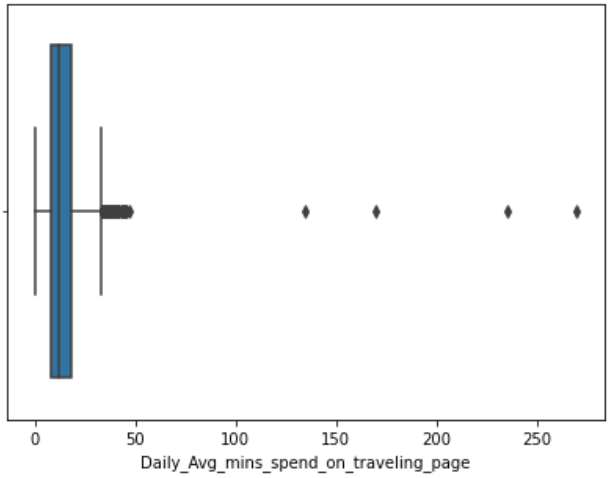






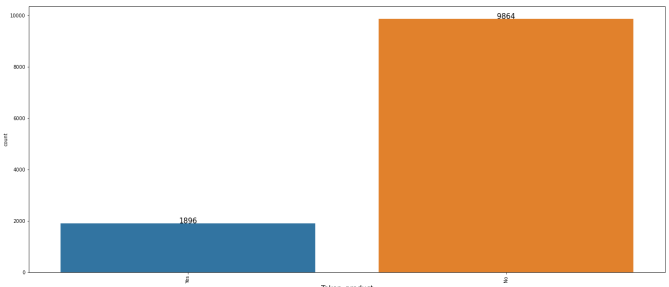


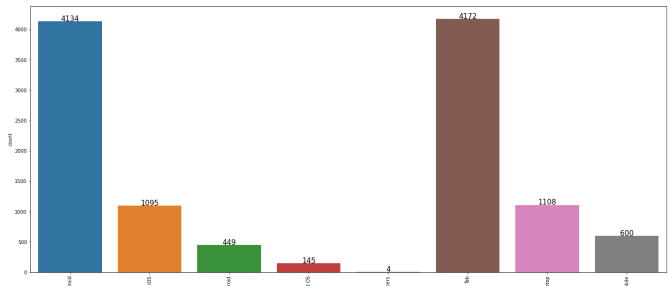


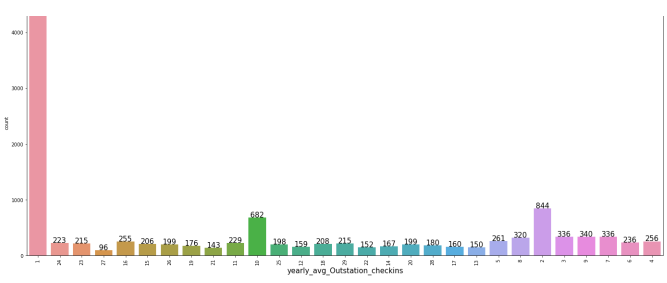


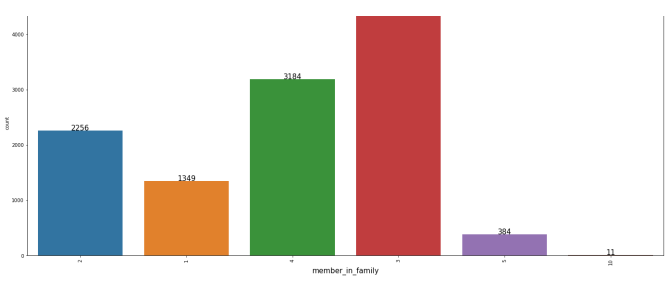
In [20]:

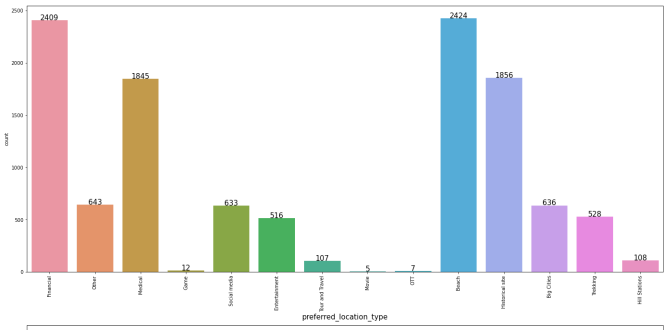
```
df_2 = df1.select_dtypes(['object','category'])

a = len(df_2.columns)     # number of rows
b = 2                     # number of columns
c = 1                    # initialize plot counter

fig = plt.figure(figsize=(a*5.8,a*12))

for i in df_2.columns:
    plt.subplot(a, b, c)
    plt.xlabel(i,fontsize =15)
    plt.xticks(rotation=90)
    ax1 = sns.countplot(df_2[i])
    c = c + 1

    for p in ax1.patches:
        ax1.annotate(format(p.get_height()), (p.get_x() + p.get_width() / 2,
                    p.get_height()), ha='center', va='center',
                    xytext=(0, 5),
                   textcoords='offset points',fontsize = 15)
```

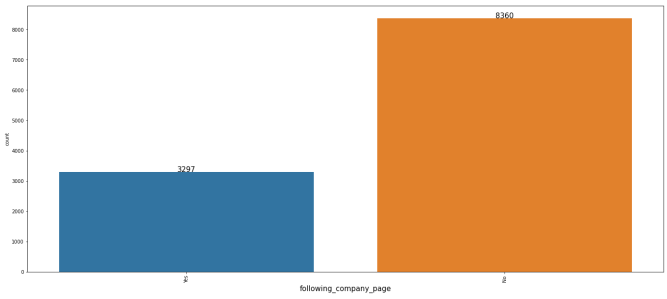


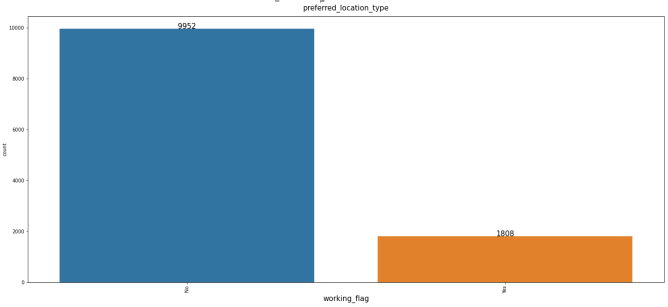


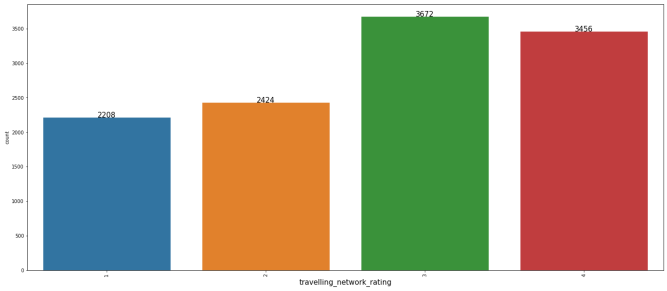


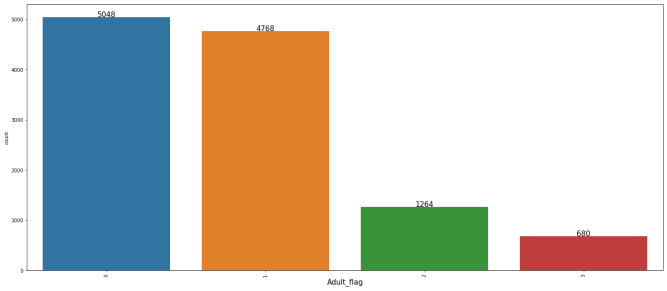






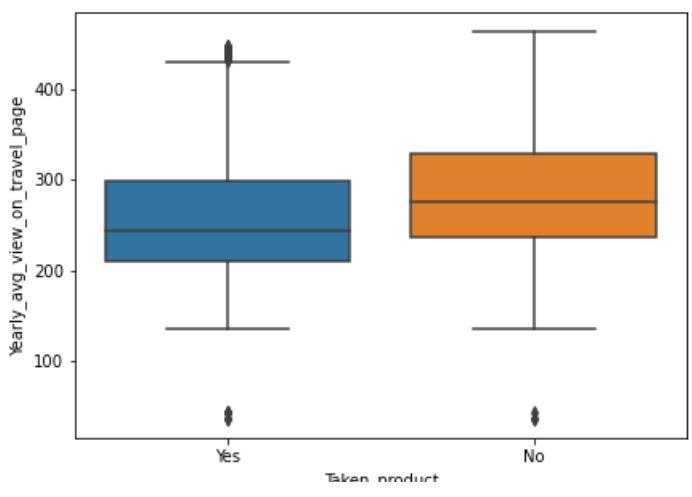


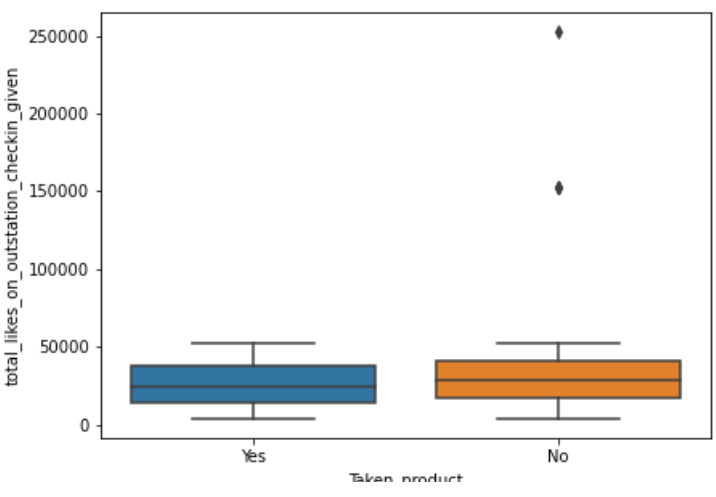


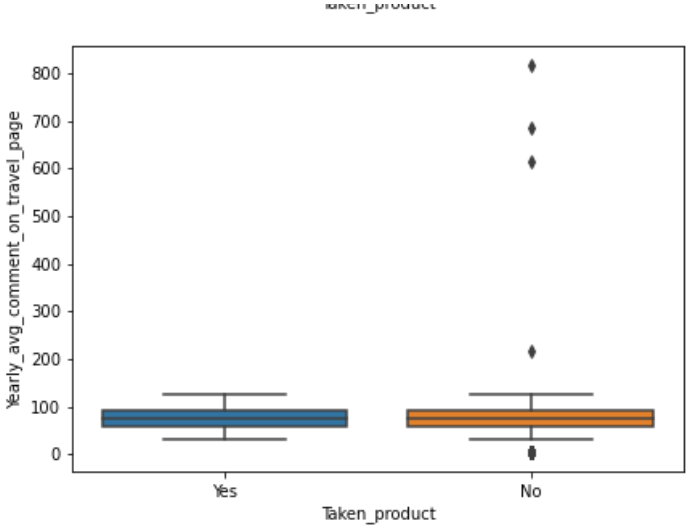


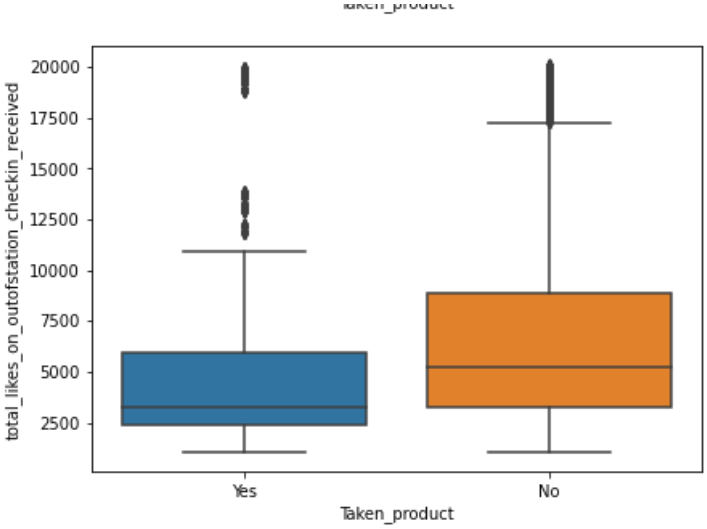
In [21]:

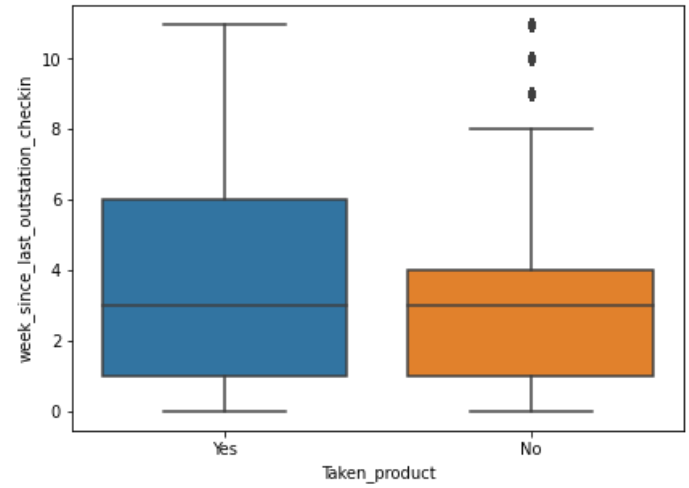
```
df_1 = df1.select_dtypes(['int64','float64'])

a = len(df_1.columns)     # number of rows
b = 2                     # number of columns
c = 1                    # initialize plot counter

fig = plt.figure(figsize=(15,40))

for i in df_1.columns:
    plt.subplot(a, b, c)
    plt.xlabel(i)
    sns.boxplot(x = df1['Taken_product'],y = df_1[i])
    c = c + 1
```

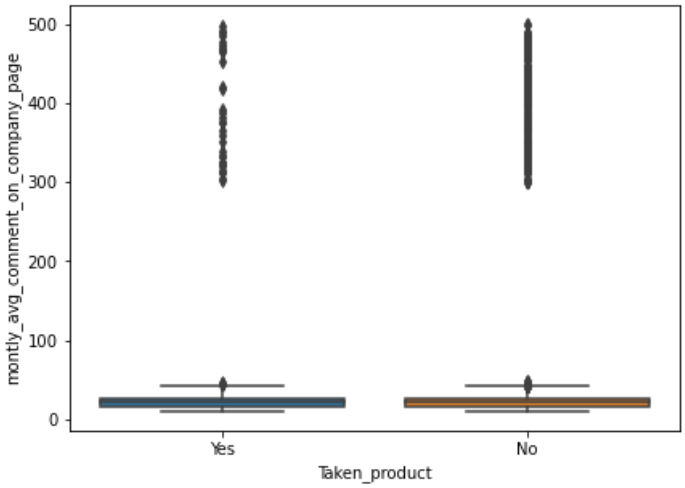


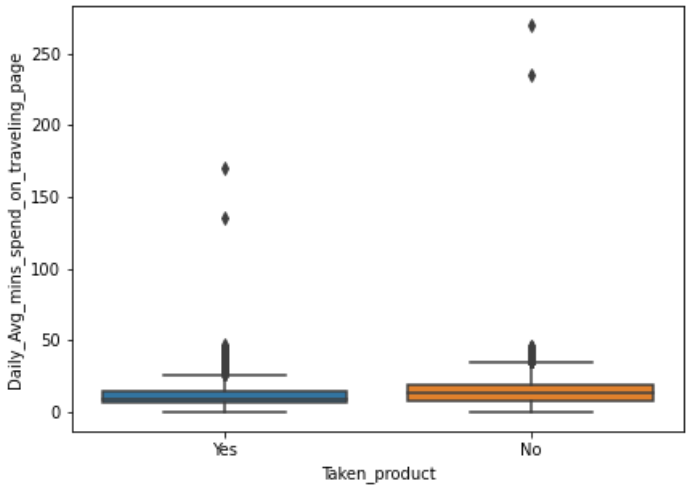








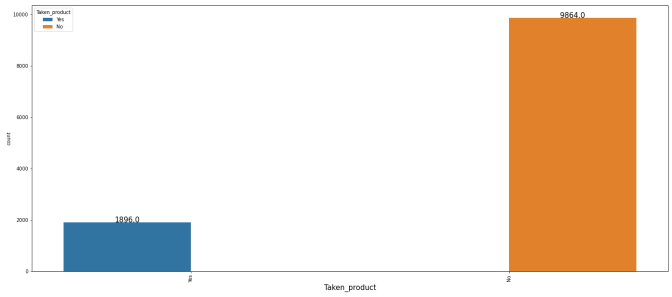


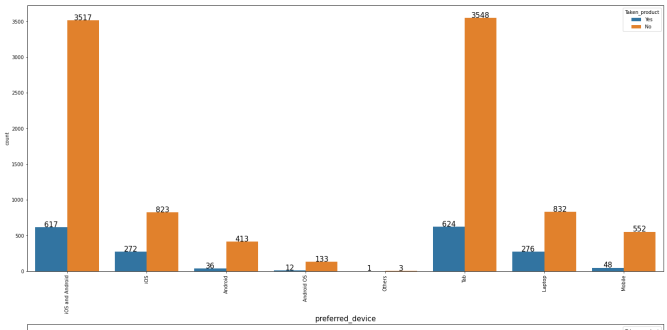


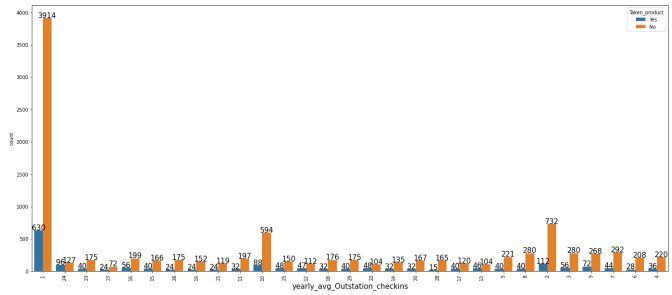
In [22]:

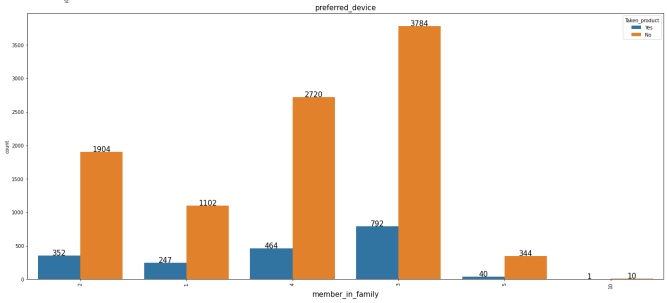
```
df_2 = df1.select_dtypes(['object','category'])

a = len(df_2.columns)     # number of rows
b = 2                     # number of columns
c = 1                    # initialize plot counter

fig = plt.figure(figsize=(a*5.8,a*12))

for i in df_2.columns:
    plt.subplot(a, b, c)
    plt.xlabel(i,fontsize =15)
    plt.xticks(rotation=90)
    ax1 = sns.countplot(df_2[i], hue=df['Taken_product'])
    c = c + 1

    for p in ax1.patches:
        ax1.annotate(format(p.get_height()), (p.get_x() + p.get_width() / 2,
                    p.get_height()), ha='center', va='center',
```

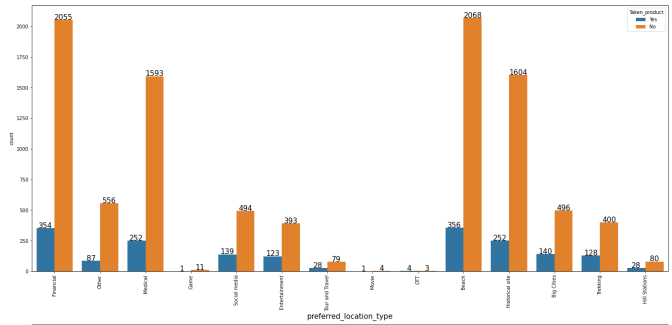
```
                xytext=(0, 5),
               textcoords='offset points', fontsize =15)
```

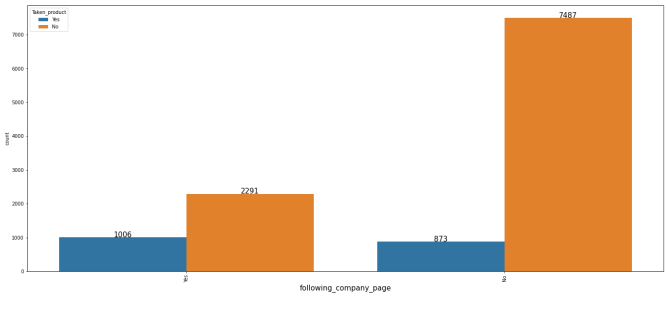


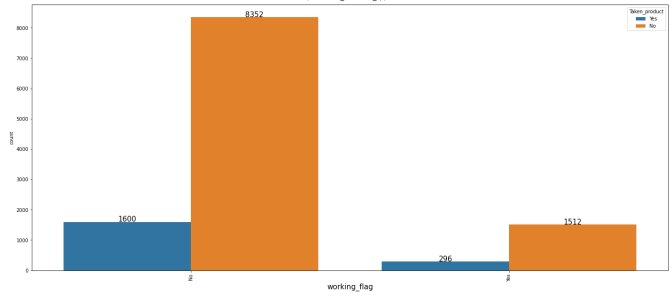


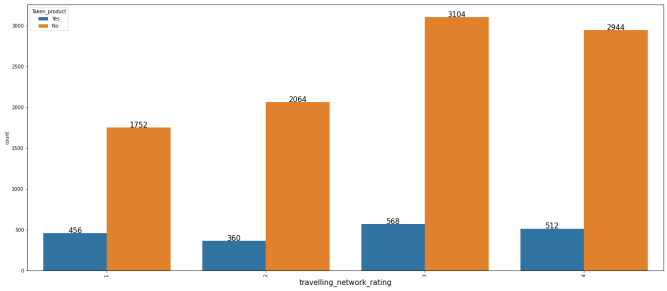


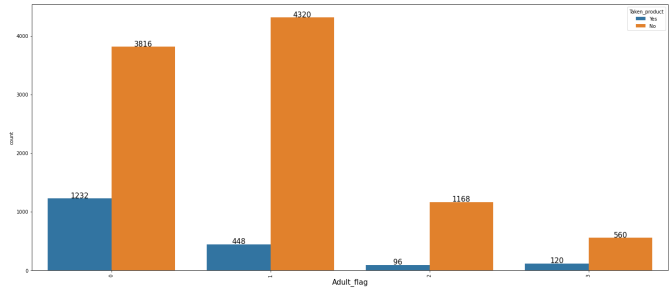






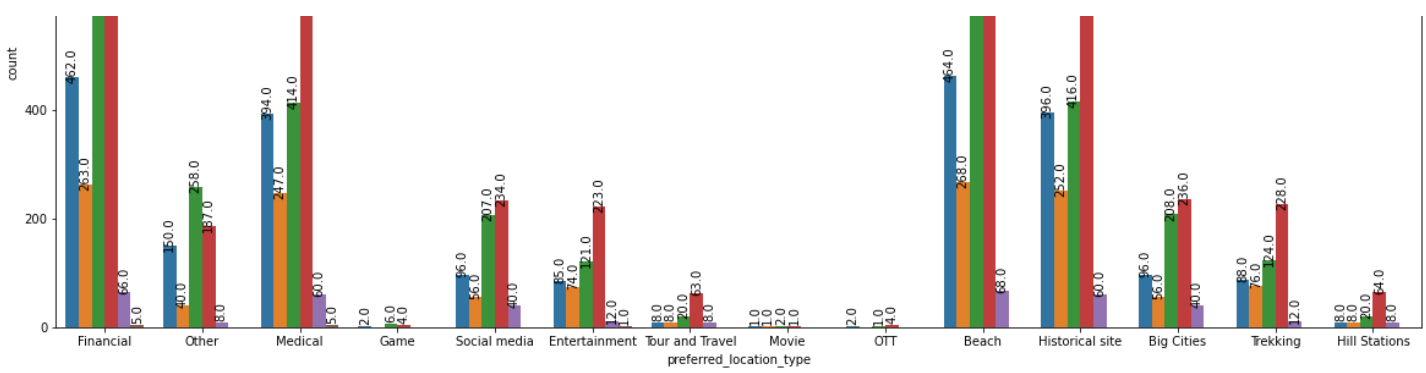






In [23]:

```
plt.figure(figsize=(20,8))
ax1 = sns.countplot(df1['preferred_location_type'],hue =df1['member_in_family'])
for p in ax1.patches:
        ax1.annotate(format(p.get_height()), (p.get_x() + p.get_width() / 2,
                p.get_height()+10), ha='center', va='center',
                xytext=(0, 5),
               textcoords='offset points',rotation =90)
```

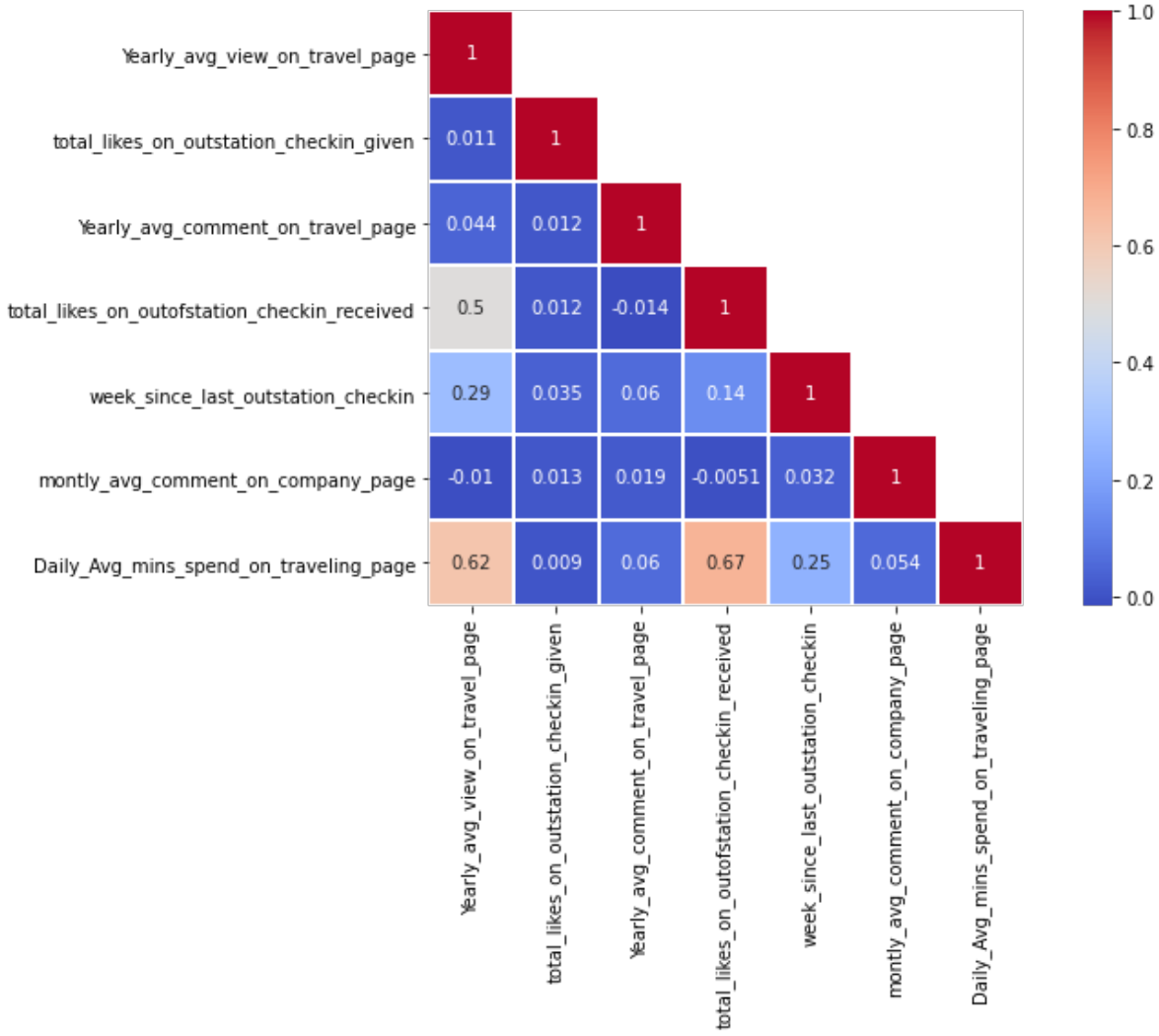


In [24]:

```
plt.figure(figsize=(12,6))

mask = np.triu(df1.corr(),k=1)
sns.heatmap(df1.corr(),cmap='coolwarm',annot=True,square=True,mask=mask,linewidths=1)
```

Out[24]:

<AxesSubplot:>



Missing Value Treatment

In [25]:

```
for col in df1.columns:
    if (df1[col].isnull().sum()>0) & (df1[col].dtype == 'O'):
        df1[col].fillna(df1[col].mode()[0],inplace= True)

    elif (df1[col].isnull().sum()>0) & (df1[col].dtype != 'O'):
        df1[col].fillna(df1[col].median(),inplace= True)
```

In [26]:

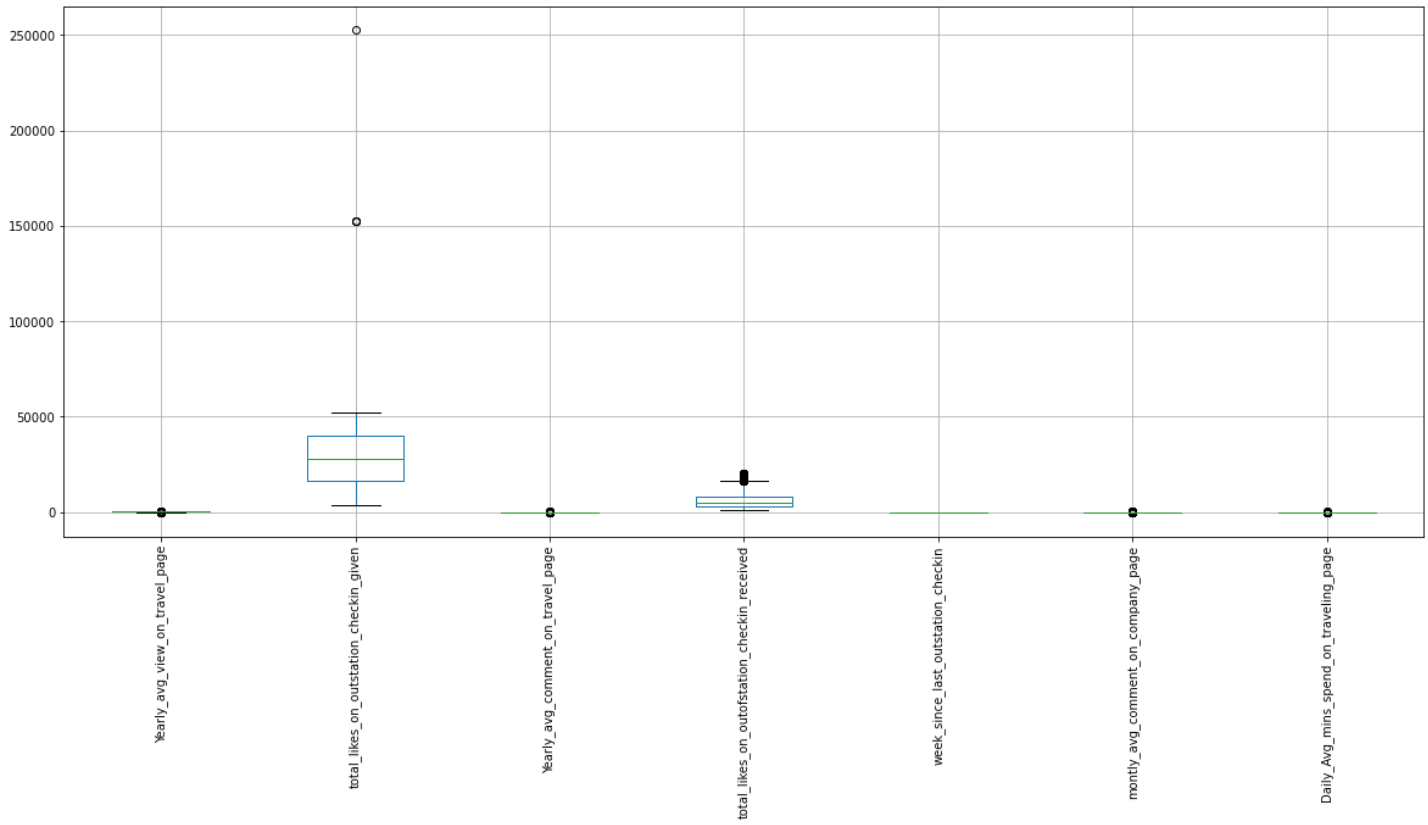
```
print('The total number of null values in the data are',df1.isnull().sum().sum())
```

```
The total number of null values in the data are 0
```

Outlier Treatment

In [27]:

```
plt.figure(figsize = (20,8))
df1.boxplot(rot = 90);
```



In [28]:

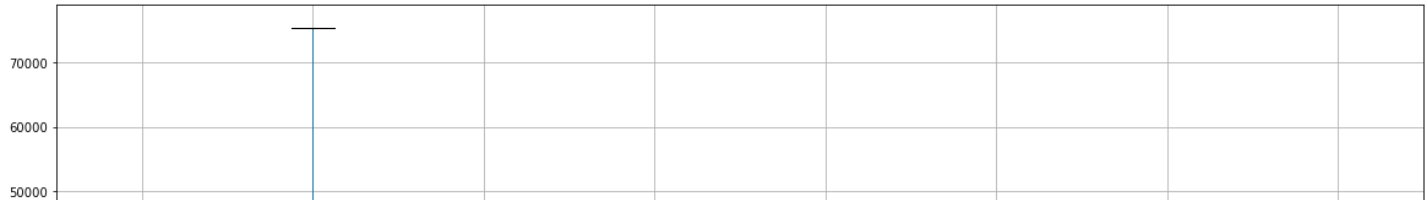
```
def remove_outlier(col):
    sorted(col)
    Q1,Q3=np.percentile(col,[25,75])
    IQR=Q3-Q1
    lr= Q1-(1.5 * IQR)
    ur= Q3+(1.5 * IQR)
    return lr, ur
```

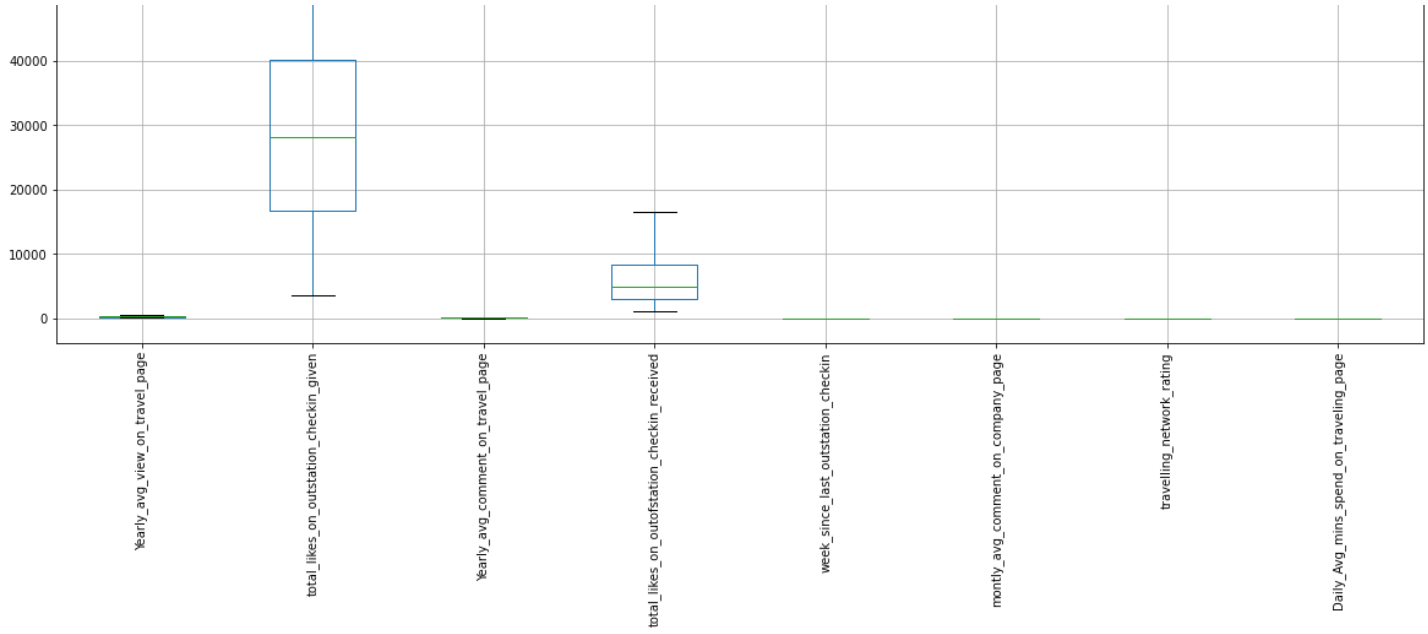
In [29]:

```
for column in df1.columns:
    if (df1[column].dtype != 'O') :
        df1[column] = df1[column].astype(int)
        lr,ur = remove_outlier(df1[column])
        df1[column]=np.where(df1[column]>ur,ur,df1[column])
        df1[column]=np.where(df1[column]<lr,lr,df1[column])
```

In [30]:

```
plt.figure(figsize = (20, 8))
df1.boxplot(rot = 90);
```





Variable Transformation / Addition of New Variables

In [31]:

```
df1['Taken_product'] = np.where(df1['Taken_product']=='Yes',1,0)
df1['Taken_product'] = df1['Taken_product'].astype('float64')
```

In [32]:

```
df1['working_flag_lbl'] = np.where(df1['working_flag']=='Yes',1,0)
df1['working_flag_lbl'] = df1['working_flag_lbl'].astype('float64')
```

In [33]:

```
df1['following_company_page_lbl'] = np.where(df1['following_company_page']=='Yes',1,0)
df1['following_company_page_lbl'] = df1['following_company_page_lbl'].astype('float64')
```

In [34]:

```
df1['Laptop/ Mobile'] = np.where(df1['preferred_device']=='Laptop','Laptop','Mobile')

df1['Laptop/Mobile_lbl'] = np.where(df1['Laptop/ Mobile']=='Laptop',0,1)
df1['Laptop/Mobile_lbl'] = df1['Laptop/Mobile_lbl'].astype('float64')
```

In [35]:

```
label  = {'Beach':14,'Financial':13,'Historical site':12,'Medical':11,'Other':10,'Big Cities':9, 'Social media':8,
          'Trekking':7, 'Entertainment':6, 'Hill Stations':5, 'Tour and Travel':4 ,'Game':3, 'OTT':2, 'Movie':1}

df1['preferred_location_type_lbl'] = df1['preferred_location_type'].apply(lambda x: label[x])
df1['preferred_location_type_lbl'] = df1['preferred_location_type_lbl'].astype('float64')
```

In [36]:

```
df1['member_in_family']               = df1['member_in_family'].astype('float64')
df1['yearly_avg_Outstation_checkins'] = df1['yearly_avg_Outstation_checkins'].astype('float64')
df1['Adult_flag']                     = df1['Adult_flag'].astype('float64')
```

In [37]:

```
df1['travelling_network_rating'] = df1['travelling_network_rating'].astype('category')
df1['travelling_network_rating'] = df1['travelling_network_rating'].cat.set_categories([4.0,3.0,2.0,1.0])
```

Removal of Unwanted Variables

In [38]:

```
df2 = df1.drop(['preferred_device','preferred_location_type','following_company_page','working_flag','Laptop/ Mobile'],axis=1)
```

In [39]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

    return(vif)
```

In [40]:

```
calc_vif(df2).sort_values(by = 'VIF', ascending = False)
```

Out[40]:

	variables	VIF
1	Yearly_avg_view_on_travel_page	27.707664
15	preferred_location_type_lbl	17.408071
8	montly_avg_comment_on_company_page	13.595628
5	Yearly_avg_comment_on_travel_page	12.837435
11	Daily_Avg_mins_spend_on_traveling_page	10.764774
14	Laptop/Mobile_lbl	8.878148
4	member_in_family	8.687737
6	total_likes_on_outofstation_checkin_received	6.864445
9	travelling_network_rating	6.858929
2	total_likes_on_outstation_checkin_given	4.935385
7	week_since_last_outstation_checkin	2.869913
3	yearly_avg_Outstation_checkins	1.927082
10	Adult_flag	1.913456
13	following_company_page_lbl	1.500105
12	working_flag_lbl	1.455599
0	Taken_product	1.346810

In [41]:

```
def calculate_vif_(X, thresh= 10.0):
    variables = list(range(X.shape[1]))
    dropped = True
    while dropped:
        dropped = False
        vif = [variance_inflation_factor(X.iloc[:, variables].values, ix)
               for ix in range(X.iloc[:, variables].shape[1])]

        maxloc = vif.index(max(vif))
        if max(vif) >= thresh:
            print('dropping \'' + X.iloc[:, variables].columns[maxloc])
            del variables[maxloc]
            dropped = True
```

```
    print('Remaining variables:')
    print(X.columns[variables])
    return X.iloc[:, variables]
```

In [42]:

```
df2 = calculate_vif_(df2)
```

```
dropping 'Yearly_avg_view_on_travel_page
dropping 'preferred_location_type_lbl
dropping 'montly_avg_comment_on_company_page
Remaining variables:
Index(['Taken_product', 'total_likes_on_outstation_checkin_given',
       'yearly_avg_Outstation_checkins', 'member_in_family',
       'Yearly_avg_comment_on_travel_page',
       'total_likes_on_outofstation_checkin_received',
       'week_since_last_outstation_checkin', 'travelling_network_rating',
       'Adult_flag', 'Daily_Avg_mins_spend_on_traveling_page',
       'working_flag_lbl', 'following_company_page_lbl', 'Laptop/Mobile_lbl'],
      dtype='object')
```

In [43]:

```
calc_vif(df2).sort_values(by = 'VIF', ascending = False)
```

Out[43]:

	variables	VIF
4	Yearly_avg_comment_on_travel_page	9.963561
9	Daily_Avg_mins_spend_on_traveling_page	8.911769
12	Laptop/Mobile_lbl	8.124978
3	member_in_family	7.706776
5	total_likes_on_outofstation_checkin_received	6.663508
7	travelling_network_rating	6.304482
1	total_likes_on_outstation_checkin_given	4.713784
6	week_since_last_outstation_checkin	2.807432
2	yearly_avg_Outstation_checkins	1.903982
8	Adult_flag	1.899842
11	following_company_page_lbl	1.493347
0	Taken_product	1.342935
10	working_flag_lbl	1.176109

In [44]:

```
print(df2['Taken_product'].value_counts())
print('\n')
print('Normalized Score is\n',df2['Taken_product'].value_counts(normalize=True))
```

```
0.0    9864
1.0    1896
Name: Taken_product, dtype: int64


Normalized Score is
 0.0    0.838776
1.0    0.161224
Name: Taken_product, dtype: float64
```

In [45]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score , roc_
curve
from imblearn.over_sampling import SMOTE

from sklearn.preprocessing import StandardScaler
```

Laptops

In [46]:

```
df3 = df2[df2['Laptop/Mobile_lbl']==0]
```

In [47]:

```
y = df3['Taken_product']
X = df3.drop('Taken_product',axis=1)
```

In [48]:

```
sc = StandardScaler()

X_fit = sc.fit_transform(X)
X = pd.DataFrame(X_fit,columns=X.columns)
```

In [49]:

```
sm = SMOTE(random_state = 42)
X_sm, y_sm = sm.fit_resample(X, y)
```

Logistic Regression

In [50]:

```
from sklearn.linear_model import LogisticRegression

X_train_LR, X_test_LR, y_train_LR, y_test_LR = train_test_split(X_sm, y_sm, test_size=0.3
, random_state=42)

model_LR = LogisticRegression()
model_LR.fit(X_train_LR, y_train_LR)
```

Out[50]:

```
LogisticRegression()
```

In [51]:

```
print('The model score for Logistic Regression training set is',model_LR.score(X_train_LR
, y_train_LR))
print('\n')
print('The model score for Logistic Regression testing set is',model_LR.score(X_test_LR,
y_test_LR))
```

```
The model score for Logistic Regression training set is 0.7491408934707904


The model score for Logistic Regression testing set is 0.752
```

In [52]:

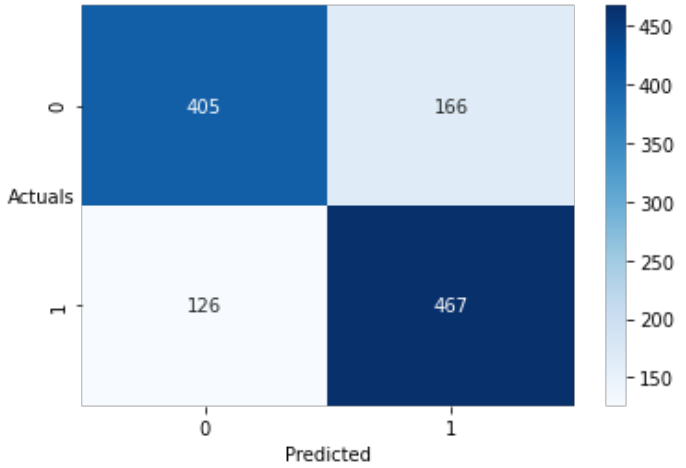
```
y_train_pred_LR = model_LR.predict(X_train_LR)
y_test_pred_LR  = model_LR.predict(X_test_LR)
```

Logistic Train Set

In [53]:

```
sns.heatmap((confusion_matrix(y_train_LR,y_train_pred_LR)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);

print('The classification report for Logistic Regression training set is\n',classificatio
n_report(y_train_LR, y_train_pred_LR))
```

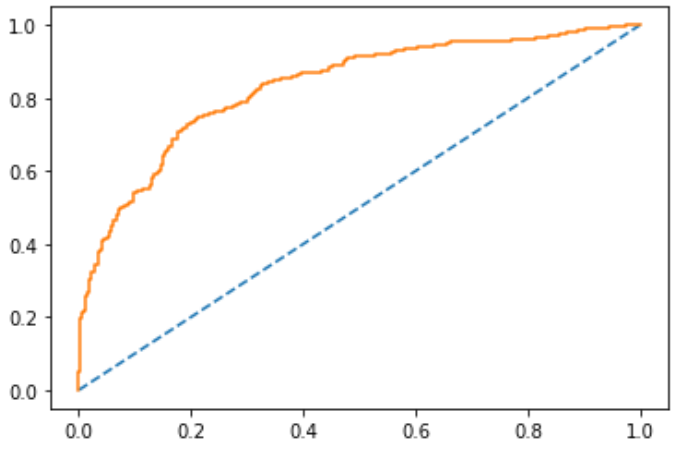
```
The classification report for Logistic Regression training set is
              precision    recall  f1-score   support

         0.0       0.76      0.71      0.74       571
         1.0       0.74      0.79      0.76       593

    accuracy                           0.75      1164
   macro avg       0.75      0.75      0.75      1164
weighted avg       0.75      0.75      0.75      1164
```



In [54]:

```
probs_LR_train = model_LR.predict_proba(X_train_LR)
probs_LR_train = probs_LR_train[:,1]
auc_train = roc_auc_score(y_train_LR, probs_LR_train)
print('The AUC score for Logistic Regression training set is: %.3f'%auc_train)

train_fpr_LR, train_tpr_LR, train_thresholds_LR = roc_curve(y_train_LR, probs_LR_train);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_LR, train_tpr_LR);
```

```
The AUC score for Logistic Regression training set is: 0.832
```



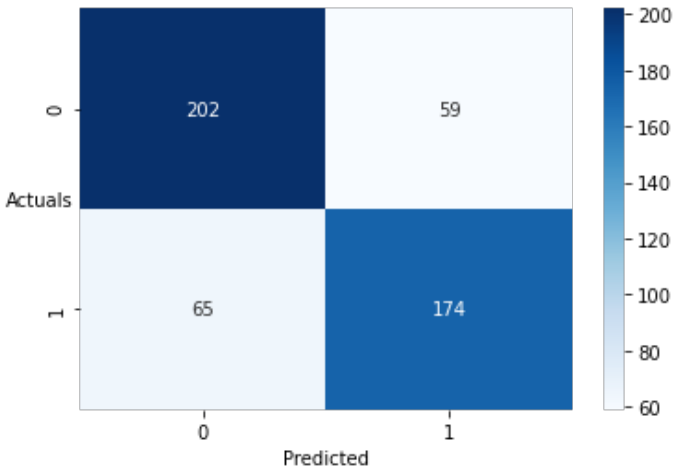
Logistic Test Set

In [55]:

```
print('The classification report for Logistic Regression testing set is\n',classification
_report(y_test_LR, y_test_pred_LR))
```

```
sns.heatmap((confusion_matrix(y_test_LR,y_test_pred_LR)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

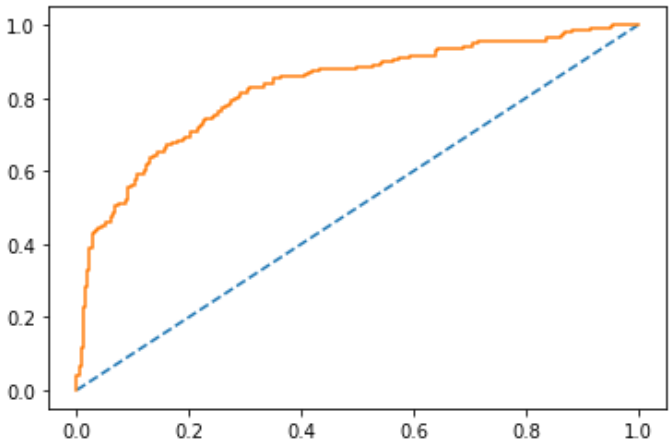
```
The classification report for Logistic Regression testing set is
              precision    recall  f1-score   support

         0.0       0.76      0.77      0.77       261
         1.0       0.75      0.73      0.74       239

    accuracy                           0.75       500
   macro avg       0.75      0.75      0.75       500
weighted avg       0.75      0.75      0.75       500
```



In [56]:

```
probs_LR_test = model_LR.predict_proba(X_test_LR)
probs_LR_test = probs_LR_test[:,1]
auc_test = roc_auc_score(y_test_LR, probs_LR_test)
print('The AUC score for Logistic Regression testing set is: %.3f'%auc_test)

test_fpr_LR, test_tpr_LR, test_thresholds_LR = roc_curve(y_test_LR, probs_LR_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_LR, test_tpr_LR);
```

```
The AUC score for Logistic Regression testing set is: 0.827
```



Linear Discriminant Analysis

In [57]:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

X_train_LDA, X_test_LDA, y_train_LDA, y_test_LDA = train_test_split(X_sm, y_sm, test_siz
e=0.3, random_state=42)
```

```
model_LDA = LinearDiscriminantAnalysis()
model_LDA.fit(X_train_LDA, y_train_LDA)
```

Out[57]:

```
LinearDiscriminantAnalysis()
```

In [58]:

```
print('The model score for Linear Discriminant Analysis training set is',model_LDA.score(
X_train_LDA, y_train_LDA))
print('\n')
print('The model score for Linear Discriminant Analysis testing set is',model_LDA.score(X
_test_LDA, y_test_LDA))
```

```
The model score for Linear Discriminant Analysis training set is 0.7542955326460481


The model score for Linear Discriminant Analysis testing set is 0.754
```

In [59]:

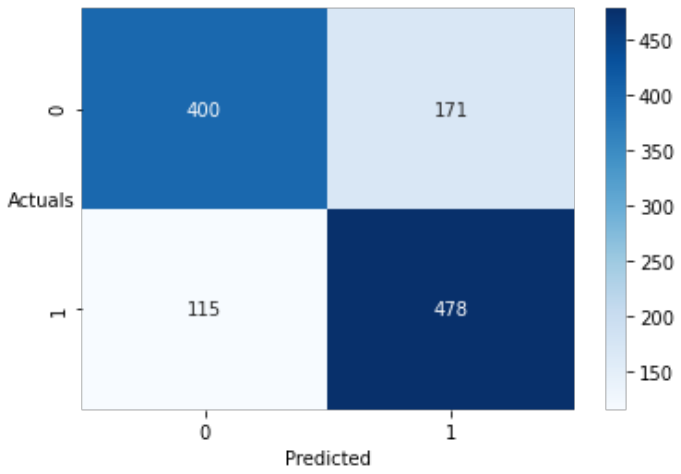
```
y_train_pred_LDA = model_LDA.predict(X_train_LDA)
y_test_pred_LDA  = model_LDA.predict(X_test_LDA)
```

Linear Discriminant Analysis Train Set

In [60]:

```
print('The classification report for Linear Discriminant Analysis training set is\n',clas
sification_report(y_train_LDA, y_train_pred_LDA))

sns.heatmap((confusion_matrix(y_train_LDA,y_train_pred_LDA)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

```
The classification report for Linear Discriminant Analysis training set is
              precision    recall  f1-score   support

         0.0       0.78      0.70      0.74       571
         1.0       0.74      0.81      0.77       593

    accuracy                           0.75      1164
   macro avg       0.76      0.75      0.75      1164
weighted avg       0.76      0.75      0.75      1164
```

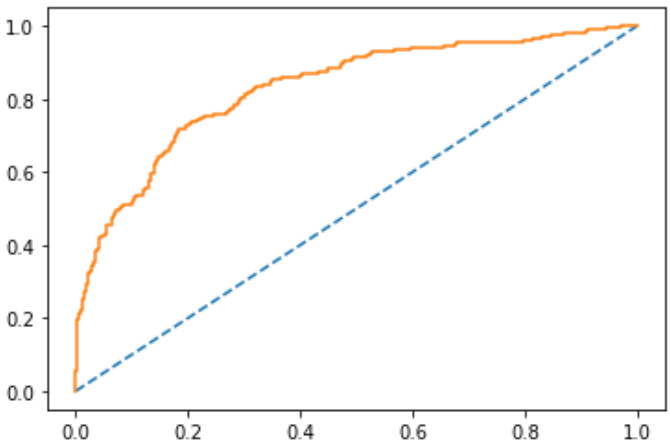


In [61]:

```
probs_LDA_train = model_LDA.predict_proba(X_train_LDA)
probs_LDA_train = probs_LDA_train[:,1]
auc_train_LDA = roc_auc_score(y_train_LDA, probs_LDA_train)
print('The AUC score for Linear Discriminant Analysis training set is: %.3f'%auc_train_LD
```

```
A)

train_fpr_LDA, train_tpr_LDA, train_thresholds_LDA = roc_curve(y_train_LDA, probs_LDA_tra
in);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_LDA, train_tpr_LDA);
```

```
The AUC score for Linear Discriminant Analysis training set is: 0.831
```

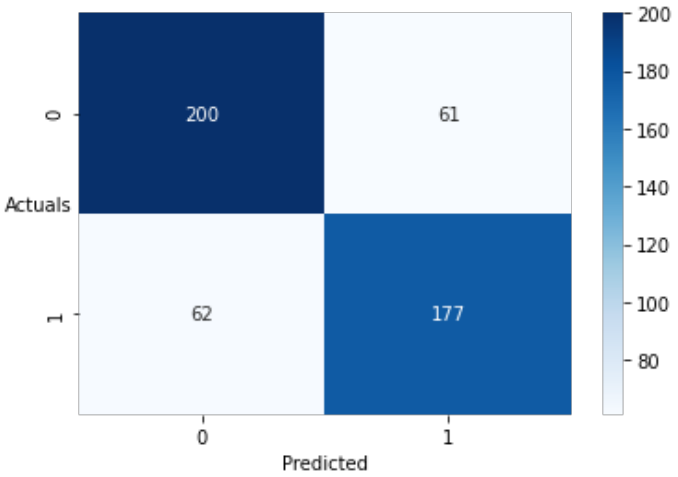


Linear Discriminant Analysis Test Set

In [62]:

```
print('The classification report for Linear Discriminant Analysis testing set is\n',class
ification_report(y_test_LDA, y_test_pred_LDA))

sns.heatmap((confusion_matrix(y_test_LDA,y_test_pred_LDA)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

```
The classification report for Linear Discriminant Analysis testing set is
              precision    recall  f1-score   support

         0.0       0.76      0.77      0.76       261
         1.0       0.74      0.74      0.74       239

    accuracy                           0.75       500
   macro avg       0.75      0.75      0.75       500
weighted avg       0.75      0.75      0.75       500
```



In [63]:

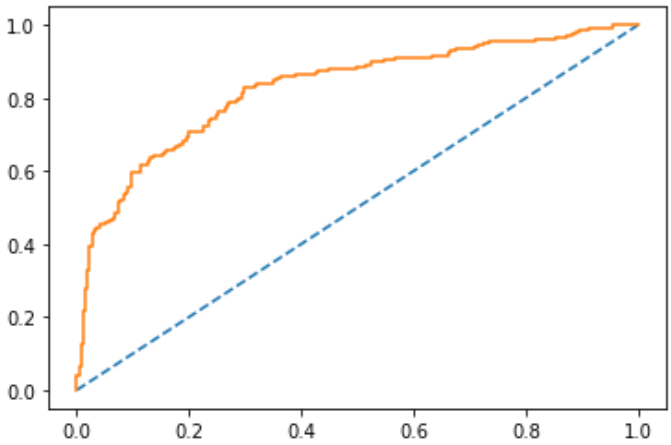
```
probs_LDA_test = model_LDA.predict_proba(X_test_LDA)
probs_LDA_test = probs_LDA_test[:,1]
auc_test_LDA = roc_auc_score(y_test_LDA, probs_LDA_test)
```

```
print('The AUC score for Linear Discriminant Analysis testing set is: %.3f'%auc_test_LDA)

test_fpr_LDA, test_tpr_LDA, test_thresholds_LDA = roc_curve(y_test_LDA, probs_LDA_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_LDA, test_tpr_LDA);
```

```
The AUC score for Linear Discriminant Analysis testing set is: 0.826
```

Out[63]:

```
[<matplotlib.lines.Line2D at 0x18e73c1b580>]
```



KNN

In [64]:

```
from sklearn.neighbors import KNeighborsClassifier

X_train_KNN, X_test_KNN, y_train_KNN, y_test_KNN = train_test_split(X_sm, y_sm, test_siz
e=0.30, random_state=42)

model_KNN = KNeighborsClassifier()
model_KNN.fit(X_train_KNN, y_train_KNN)
```

Out[64]:

```
KNeighborsClassifier()
```

In [65]:

```
print('The model score for KNN training set is',model_KNN.score(X_train_KNN, y_train_KNN)
)
print('\n')
print('The model score for KNN testing set is',model_KNN.score(X_test_KNN, y_test_KNN))
```

```
The model score for KNN training set is 0.9819587628865979


The model score for KNN testing set is 0.95
```

In [66]:

```
y_train_pred_KNN = model_KNN.predict(X_train_KNN)
y_test_pred_KNN  = model_KNN.predict(X_test_KNN)
```

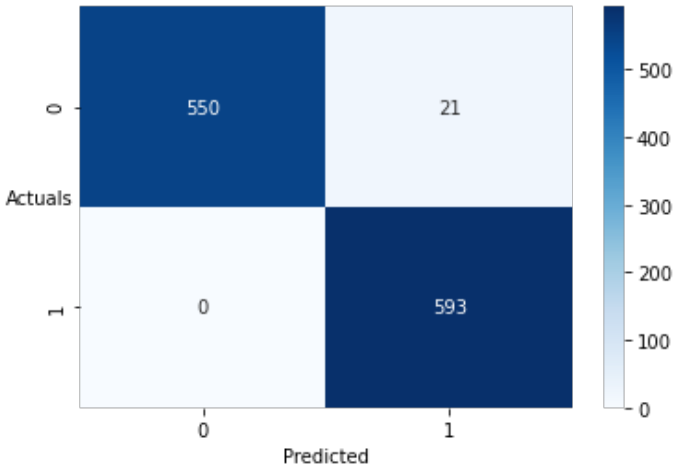
KNN Train Set

In [67]:

```
print('The classification report for KNN set is\n',classification_report(y_train_KNN, y_t
rain_pred_KNN))

sns.heatmap((confusion_matrix(y_train_KNN,y_train_pred_KNN)),annot=True,fmt='.5g'
            ,cmap='Blues');
```

```
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

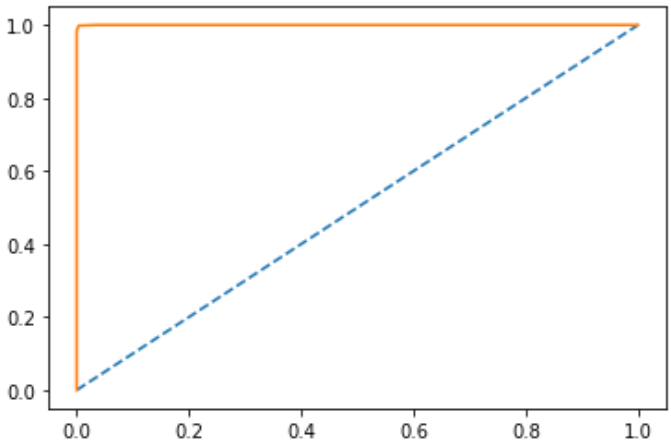
```
The classification report for KNN set is
              precision    recall  f1-score   support

         0.0       1.00      0.96      0.98       571
         1.0       0.97      1.00      0.98       593

    accuracy                           0.98      1164
   macro avg       0.98      0.98      0.98      1164
weighted avg       0.98      0.98      0.98      1164
```



In [68]:

```
probs_KNN_train = model_KNN.predict_proba(X_train_KNN)
probs_KNN_train = probs_KNN_train[:,1]
auc_train_KNN = roc_auc_score(y_train_KNN, probs_KNN_train)

print('The AUC score for KNN training set is: %.3f'%auc_train_KNN)

train_fpr_KNN, train_tpr_KNN, train_thresholds_KNN = roc_curve(y_train_KNN, probs_KNN_tra
in);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_KNN, train_tpr_KNN);
```

```
The AUC score for KNN training set is: 1.000
```



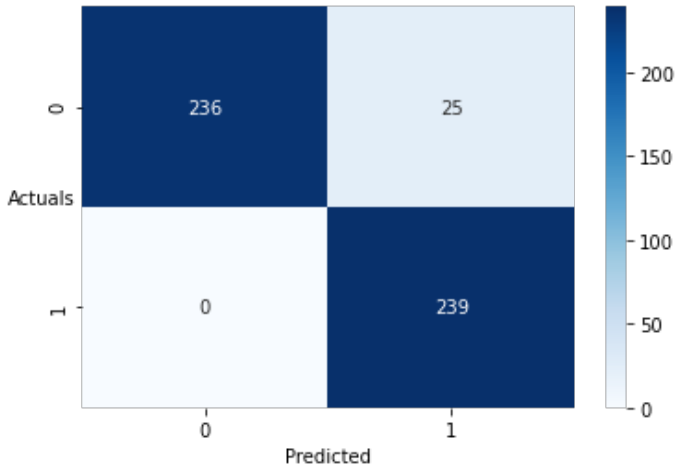
KNN Test Set

In [69]:

```
print('The classification report for KNN testing set is\n',classification_report(y_test_K
NN, y_test_pred_KNN))

sns.heatmap((confusion_matrix(y_test_KNN,y_test_pred_KNN)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
```

```
plt.ylabel('Actuals',rotation=0);
```

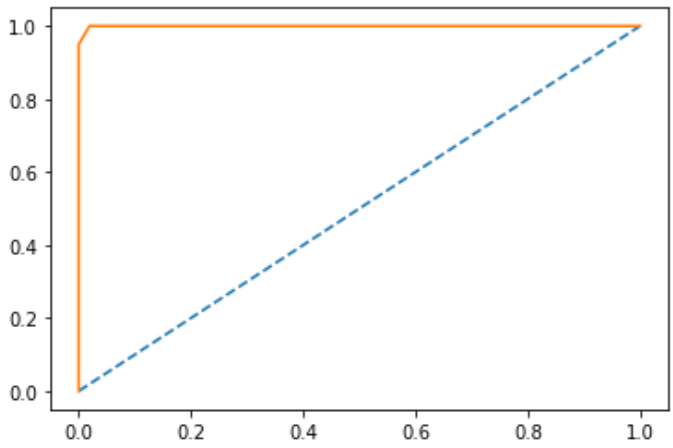
```
The classification report for KNN testing set is
              precision    recall  f1-score   support

         0.0       1.00      0.90      0.95       261
         1.0       0.91      1.00      0.95       239

    accuracy                           0.95       500
   macro avg       0.95      0.95      0.95       500
weighted avg       0.95      0.95      0.95       500
```



In [70]:

```
probs_KNN_test = model_KNN.predict_proba(X_test_KNN)
probs_KNN_test = probs_KNN_test[:,1]
auc_test_KNN = roc_auc_score(y_test_KNN, probs_KNN_test)

print('The AUC score for KNN testing set is: %.3f'%auc_test_KNN)

test_fpr_KNN, test_tpr_KNN, test_thresholds_KNN = roc_curve(y_test_KNN, probs_KNN_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_KNN, test_tpr_KNN);
```

```
The AUC score for KNN testing set is: 1.000
```



Naive Bayes Model

In [71]:

```
from sklearn.naive_bayes import GaussianNB

X_train_NB, X_test_NB, y_train_NB, y_test_NB = train_test_split(X_sm, y_sm, test_size=0.3
0, random_state=42)

model_NB = GaussianNB()
model_NB.fit(X_train_NB, y_train_NB)
```

Out[71]:

```
GaussianNB()
```

In [72]:

```
print('The model score for Naive Bayes Model training set is',model_NB.score(X_train_NB,
y_train_NB))
print('\n')
print('The model score for Naive Bayes Model testing set is',model_NB.score(X_test_NB, y_
test_NB))
```

```
The model score for Naive Bayes Model training set is 0.718213058419244


The model score for Naive Bayes Model testing set is 0.73
```

In [73]:

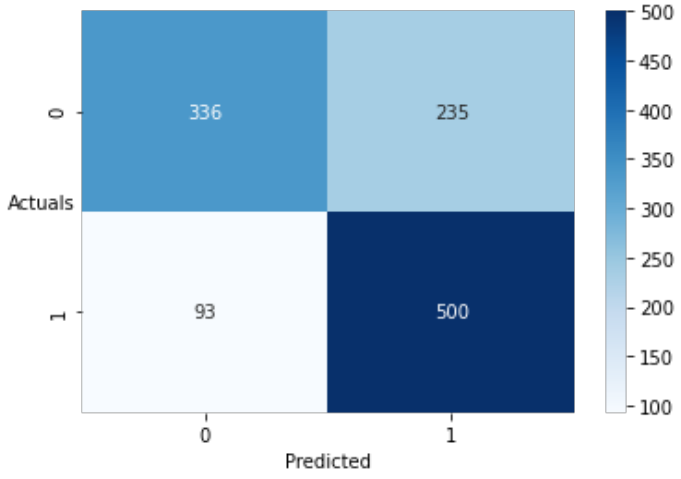
```
y_train_pred_NB = model_NB.predict(X_train_NB)
y_test_pred_NB  = model_NB.predict(X_test_NB)
```

Naive Bayes Train Set

In [74]:

```
print('The classification report for Naive Bayes Model set is\n',classification_report(y_
train_NB, y_train_pred_NB))

sns.heatmap((confusion_matrix(y_train_NB,y_train_pred_NB)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

```
The classification report for Naive Bayes Model set is
              precision    recall  f1-score   support

         0.0       0.78      0.59      0.67       571
         1.0       0.68      0.84      0.75       593

    accuracy                           0.72      1164
   macro avg       0.73      0.72      0.71      1164
weighted avg       0.73      0.72      0.71      1164
```

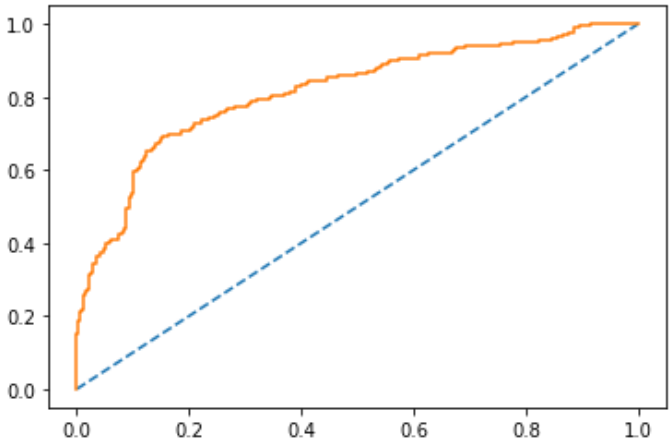


In [75]:

```
probs_NB_train = model_NB.predict_proba(X_train_NB)
probs_NB_train = probs_NB_train[:,1]
auc_train_NB = roc_auc_score(y_train_NB, probs_NB_train)

print('The AUC score for Naive Bayes training set is: %.3f'%auc_train_NB)

train_fpr_NB, train_tpr_NB, train_thresholds_NB = roc_curve(y_train_NB, probs_NB_train);
```

```
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_NB, train_tpr_NB);
```

```
The AUC score for Naive Bayes training set is: 0.816
```

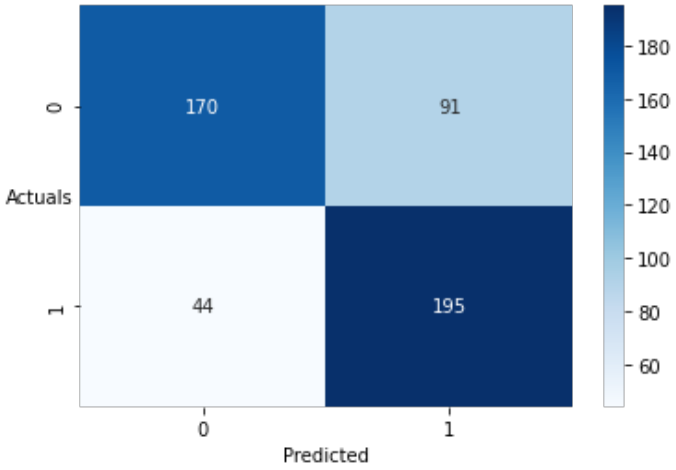


Naive Bayes Test Set

In [76]:

```
print('The classification report for Naive bayes Model testing set is\n',classification_r
eport(y_test_NB, y_test_pred_NB))

sns.heatmap((confusion_matrix(y_test_NB,y_test_pred_NB)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

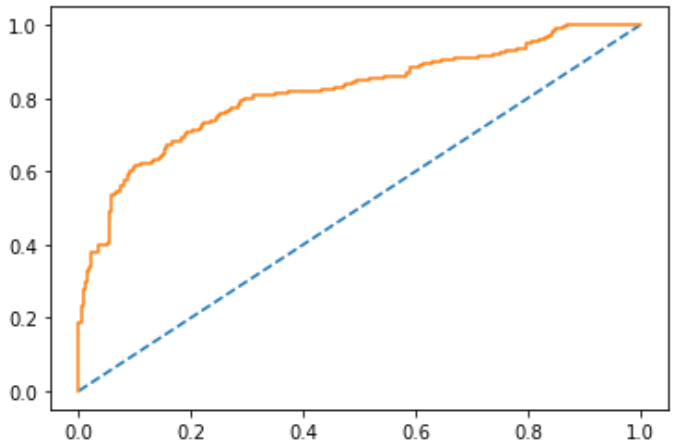
```
The classification report for Naive bayes Model testing set is
              precision    recall  f1-score   support

         0.0       0.79      0.65      0.72       261
         1.0       0.68      0.82      0.74       239

    accuracy                           0.73       500
   macro avg       0.74      0.73      0.73       500
weighted avg       0.74      0.73      0.73       500
```



In [77]:

```
probs_NB_test = model_NB.predict_proba(X_test_NB)
probs_NB_test = probs_NB_test[:,1]
auc_test_NB   = roc_auc_score(y_test_NB, probs_NB_test)

print('The AUC score for Naive Bayes testing set is: %.3f'%auc_test_NB)

test_fpr_NB, test_tpr_NB, test_thresholds_NB = roc_curve(y_test_NB, probs_NB_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_NB, test_tpr_NB);
```

```
The AUC score for Naive Bayes testing set is: 0.816
```



Decision Tree Classifier

In [78]:

```
from sklearn.tree import DecisionTreeClassifier

X_train_DT, X_test_DT, y_train_DT, y_test_DT = train_test_split(X_sm, y_sm, test_size=0.3
, random_state=42)

model_DT = DecisionTreeClassifier()
model_DT.fit(X_train_DT, y_train_DT)
```

Out[78]:

```
DecisionTreeClassifier()
```

In [79]:

```
print('The model score for Decision Tree Classifier training set is',model_DT.score(X_tra
in_DT,y_train_DT))
print('\n')
print('The model score for Decision Tree Classifier testing set is',model_DT.score(X_test
_DT,y_test_DT))
```

```
The model score for Decision Tree Classifier training set is 1.0


The model score for Decision Tree Classifier testing set is 0.958
```

In [80]:

```
y_train_pred_DT = model_DT.predict(X_train_DT)
y_test_pred_DT  = model_DT.predict(X_test_DT)
```

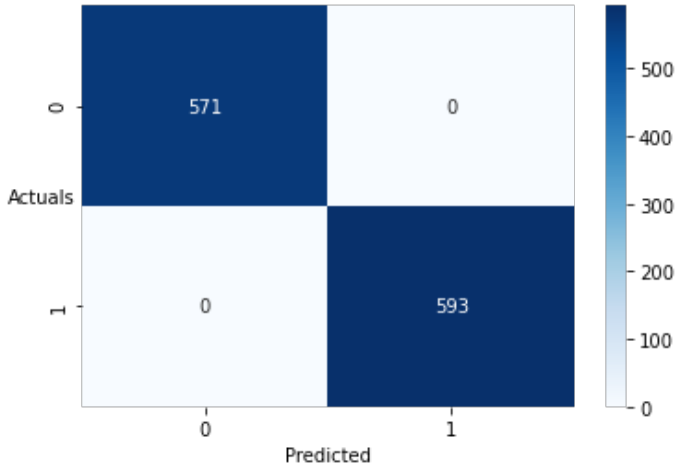
Decision Tree Classifier Train Set

In [81]:

```
print('The classification report for Decision Tree training set is\n',classification_repo
rt(y_train_DT, y_train_pred_DT))

sns.heatmap((confusion_matrix(y_train_DT,y_train_pred_DT)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

```
The classification report for Decision Tree training set is
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       571
         1.0       1.00      1.00      1.00       593
```

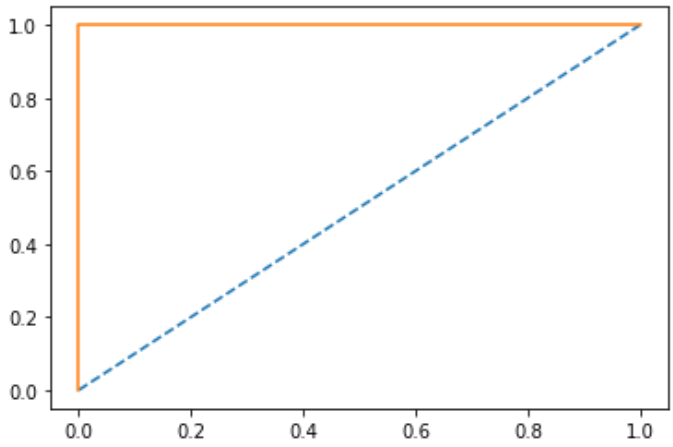
```
    accuracy                           1.00      1164
   macro avg       1.00      1.00      1.00      1164
weighted avg       1.00      1.00      1.00      1164
```



In [82]:

```
probs_DT_train = model_DT.predict_proba(X_train_DT)
probs_DT_train = probs_DT_train[:,1]
auc_train_DT = roc_auc_score(y_train_DT, probs_DT_train)
print('The AUC score for Decision Tree training set is: %.3f'%auc_train_DT)

train_fpr_DT, train_tpr_DT, train_thresholds_DT = roc_curve(y_train_DT, probs_DT_train);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_DT, train_tpr_DT);
```

```
The AUC score for Decision Tree training set is: 1.000
```



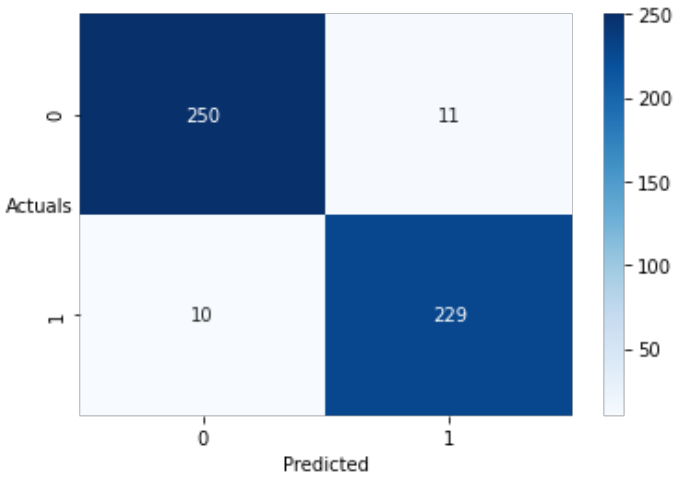
Decision Tree Classifier Test Set

In [83]:

```
print('The classification report for Decision Tree testing set is\n',classification_repor
t(y_test_DT, y_test_pred_DT))

sns.heatmap((confusion_matrix(y_test_DT,y_test_pred_DT)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

```
The classification report for Decision Tree testing set is
              precision    recall  f1-score   support

         0.0       0.96      0.96      0.96       261
         1.0       0.95      0.96      0.96       239

    accuracy                           0.96       500
   macro avg       0.96      0.96      0.96       500
```

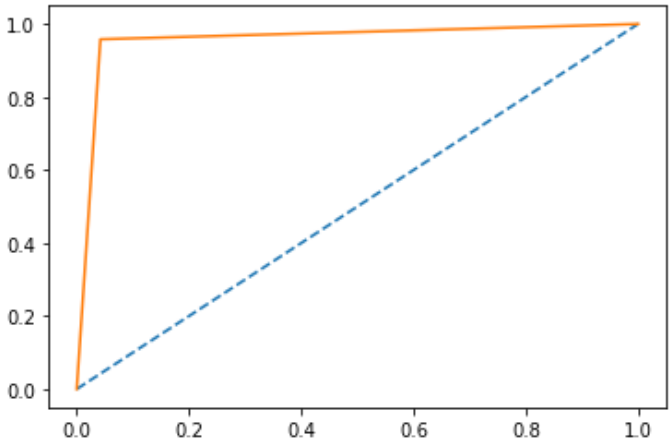
weighted avg 0.96 0.96 0.96 500



In [84]:

```
probs_DT_test = model_DT.predict_proba(X_test_DT)
probs_DT_test = probs_DT_test[:,1]
auc_test_DT = roc_auc_score(y_test_DT, probs_DT_test)
print('The AUC score for Decision Tree testing set is: %.3f'%auc_test_DT)

test_fpr_DT, test_tpr_DT, test_thresholds_DT = roc_curve(y_test_DT, probs_DT_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_DT, test_tpr_DT);
```

The AUC score for Decision Tree testing set is: 0.958



Random Forest Classifier

In [85]:

```
from sklearn.ensemble import RandomForestClassifier

X_train_RFC, X_test_RFC, y_train_RFC, y_test_RFC = train_test_split(X_sm, y_sm, test_siz
e=0.3, random_state=42)

rfcl = RandomForestClassifier()
rfcl = rfcl.fit(X_train_RFC, y_train_RFC)
```

In [86]:

```
print('The model score for Random Forest Classifier training set is',rfcl.score(X_train_R
FC,y_train_RFC))
print('\n')
print('The model score for Random Forest Classifier testing set is',rfcl.score(X_test_RFC
,y_test_RFC))
```

The model score for Random Forest Classifier training set is 1.0

```
The model score for Random Forest Classifier testing set is 0.988
```

In [87]:

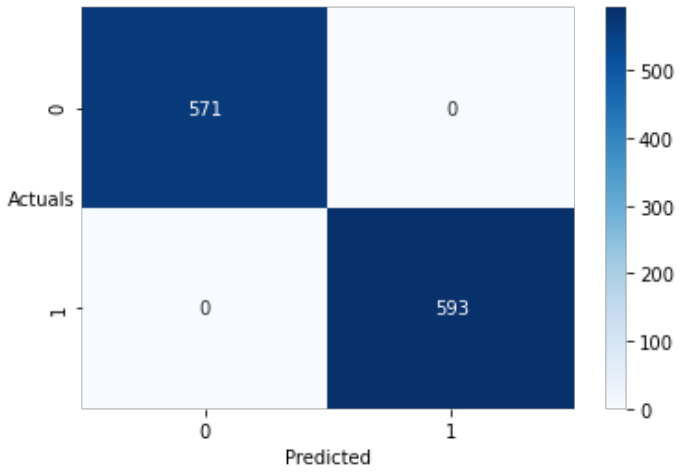
```
y_train_pred_RFC = rfcl.predict(X_train_RFC)
y_test_pred_RFC  = rfcl.predict(X_test_RFC)
```

Random Forest Classifier on Train Set

In [88]:

```
print('The classification report for RFC training set is\n',classification_report(y_train
_RFC, y_train_pred_RFC))

sns.heatmap((confusion_matrix(y_train_RFC,y_train_pred_RFC)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

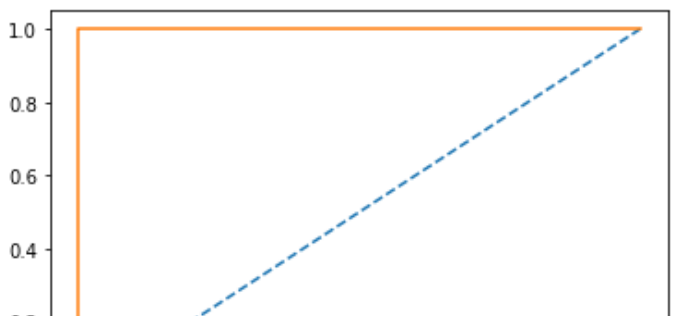
```
The classification report for RFC training set is
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       571
         1.0       1.00      1.00      1.00       593

    accuracy                           1.00      1164
   macro avg       1.00      1.00      1.00      1164
weighted avg       1.00      1.00      1.00      1164
```

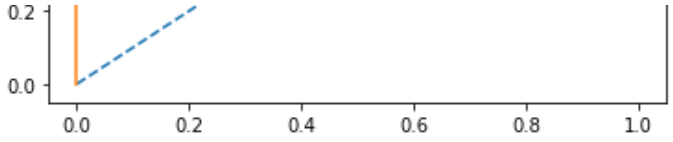


In [89]:

```
probs_RFC_train = rfcl.predict_proba(X_train_RFC)
probs_RFC_train = probs_RFC_train[:,1]
auc_train_RFC = roc_auc_score(y_train_RFC, probs_RFC_train)
print('The AUC score for RFC training set is: %.3f'%auc_train_RFC)

train_fpr_RFC, train_tpr_RFC, train_thresholds_RFC = roc_curve(y_train_RFC, probs_RFC_tra
in);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_RFC, train_tpr_RFC);
```

```
The AUC score for RFC training set is: 1.000
```



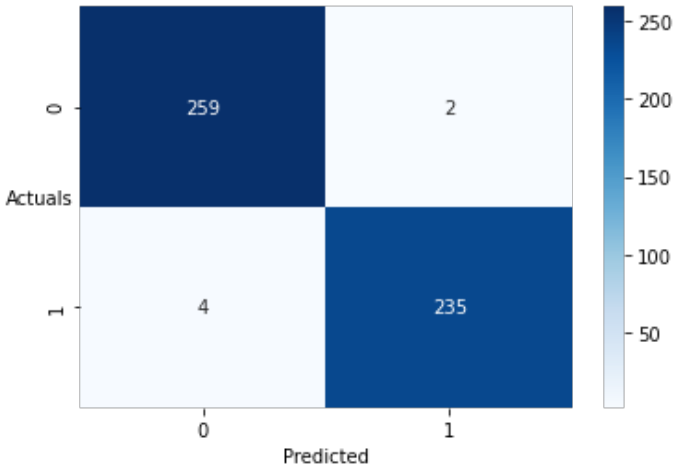


Random Forest Classifier on Test Set

In [90]:

```
print('The classification report for RFC testing set is\n',classification_report(y_test_RFC, y_test_pred_RFC))

sns.heatmap((confusion_matrix(y_test_RFC,y_test_pred_RFC)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

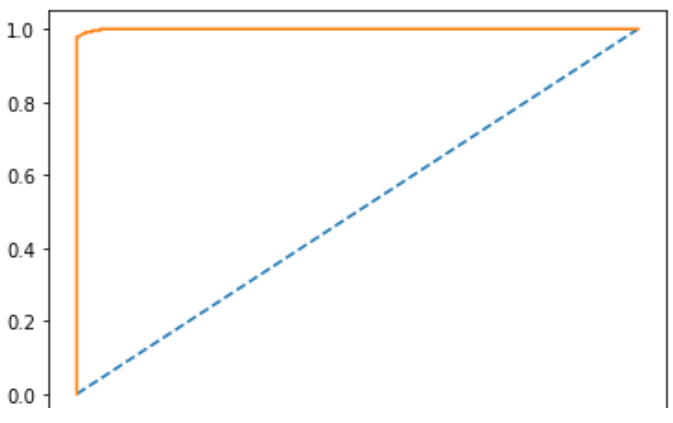
```
The classification report for RFC testing set is
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.99       261
         1.0       0.99      0.98      0.99       239

    accuracy                           0.99       500
   macro avg       0.99      0.99      0.99       500
weighted avg       0.99      0.99      0.99       500
```



In [91]:

```
probs_RFC_test = rfcl.predict_proba(X_test_RFC)
probs_RFC_test = probs_RFC_test[:,1]
auc_test_RFC = roc_auc_score(y_test_RFC, probs_RFC_test)
print('The AUC score for RFC testing set is: %.3f'%auc_test_RFC)

test_fpr_RFC, test_tpr_RFC, test_thresholds_RFC = roc_curve(y_test_RFC, probs_RFC_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_RFC, test_tpr_RFC);
```

```
The AUC score for RFC testing set is: 1.000
```



Model Tuning

Bagging

In [92]:

```
from sklearn.ensemble import BaggingClassifier

X_train_bg, X_test_bg, y_train_bg, y_test_bg = train_test_split(X_sm, y_sm, test_size=0.3
, random_state=42)

bgcl = BaggingClassifier(n_estimators=50,random_state=1)
bgcl = bgcl.fit(X_train_bg, y_train_bg)
```

In [93]:

```
print('The model score for Bagging training set is',bgcl.score(X_train_bg,y_train_bg))
print('\n')
print('The model score for Bagging testing set is',bgcl.score(X_test_bg,y_test_bg))
```

```
The model score for Bagging training set is 1.0


The model score for Bagging testing set is 0.97
```

In [94]:

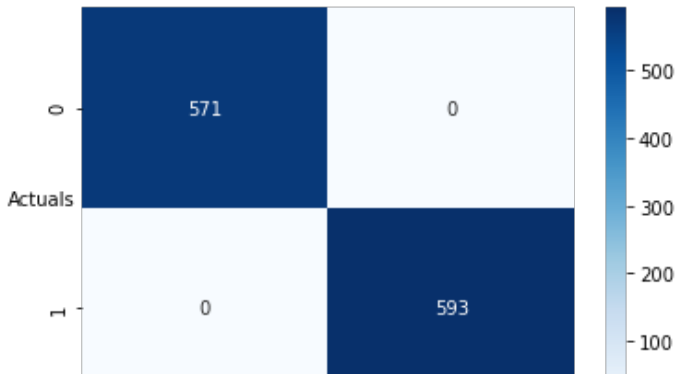
```
y_train_pred_bg = bgcl.predict(X_train_bg)
y_test_pred_bg  = bgcl.predict(X_test_bg)
```

Bagging on Train Test

In [95]:

```
print('The classification report for Bagging training set is\n',classification_report(y_t
rain_bg, y_train_pred_bg))

sns.heatmap((confusion_matrix(y_train_bg,y_train_pred_bg)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

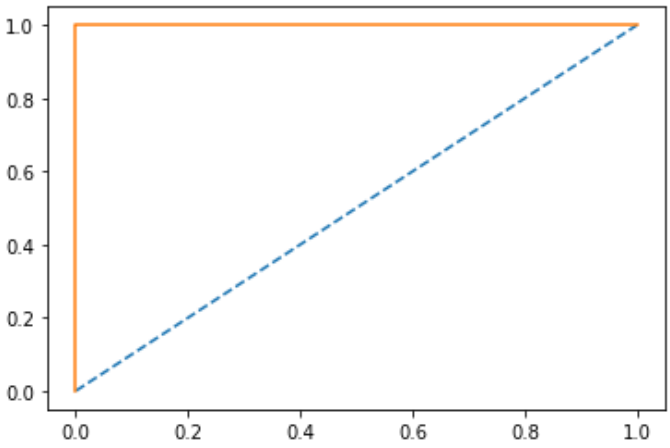
```
The classification report for Bagging training set is
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       571
         1.0       1.00      1.00      1.00       593

    accuracy                           1.00      1164
   macro avg       1.00      1.00      1.00      1164
weighted avg       1.00      1.00      1.00      1164
```



In [96]:

```
probs_bg_train = bgcl.predict_proba(X_train_bg)
probs_bg_train = probs_bg_train[:,1]
auc_train_bg = roc_auc_score(y_train_bg, probs_bg_train)
print('The AUC score for Bagging training set is: %.3f'%auc_train_bg)

train_fpr_bg, train_tpr_bg, train_thresholds_bg = roc_curve(y_train_bg, probs_bg_train);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_bg, train_tpr_bg);
```

```
The AUC score for Bagging training set is: 1.000
```

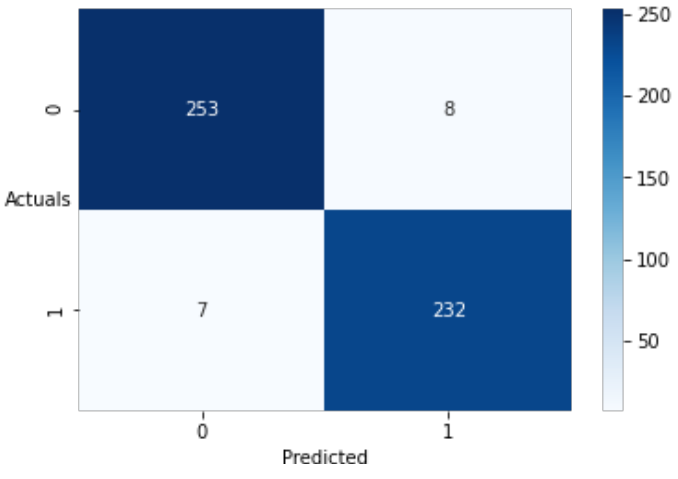


Bagging on Test Set

In [97]:

```
print('The classification report for Bagging testing set is\n',classification_report(y_te
st_bg, y_test_pred_bg))

sns.heatmap((confusion_matrix(y_test_bg,y_test_pred_bg)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

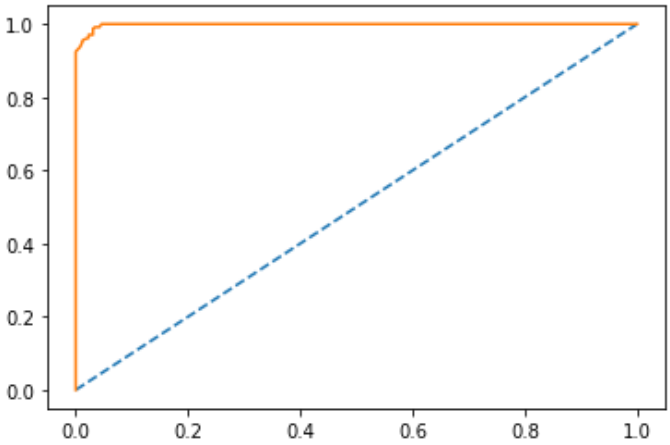
```
The classification report for Bagging testing set is
              precision    recall  f1-score   support

         0.0       0.97      0.97      0.97       261
         1.0       0.97      0.97      0.97       239

    accuracy                           0.97       500
   macro avg       0.97      0.97      0.97       500
weighted avg       0.97      0.97      0.97       500
```



In [98]:

```
probs_bg_test = bgcl.predict_proba(X_test_bg)
probs_bg_test = probs_bg_test[:,1]
auc_test_bg = roc_auc_score(y_test_bg, probs_bg_test)
print('The AUC score for Bagging testing set is: %.3f'%auc_test_bg)

test_fpr_bg, test_tpr_bg, test_thresholds_bg = roc_curve(y_test_bg, probs_bg_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_bg, test_tpr_bg);
```

```
The AUC score for Bagging testing set is: 0.998
```



AdaBoosting

In [99]:

```
from sklearn.ensemble import AdaBoostClassifier

X_train_adb, X_test_adb, y_train_adb, y_test_adb = train_test_split(X_sm, y_sm, test_siz
e=0.3, random_state=42)

abcl = AdaBoostClassifier(n_estimators=50,random_state=1)
abcl = abcl.fit(X_train_adb, y_train_adb)
```

In [100]:

```
print('The model score for AdaBoosting training set is',abcl.score(X_train_adb,y_train_ad
b))
print('\n')
print('The model score for AdaBoosting testing set is',abcl.score(X_test_adb,y_test_adb))
```

```
The model score for AdaBoosting training set is 0.8556701030927835


The model score for AdaBoosting testing set is 0.832
```

In [101]:

```
y_train_pred_adb = abcl.predict(X_train_adb)
y_test_pred_adb  = abcl.predict(X_test_adb)
```

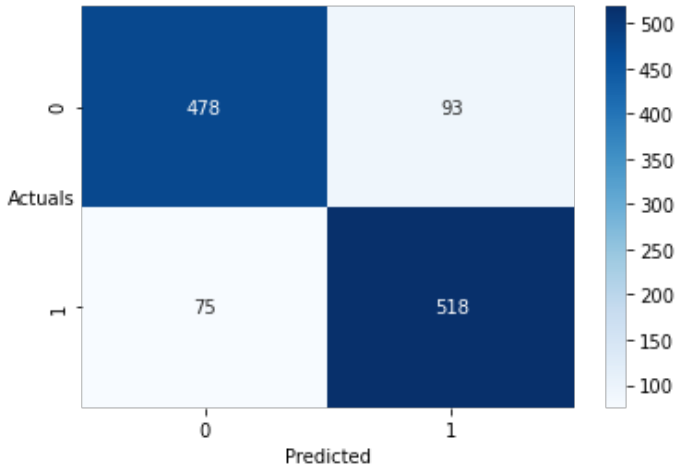
AdaBoosting on Train Set

In [102]:

```
print('The classification report for Adaboosting training set is\n',classification_report
(y_train_adb, y_train_pred_adb))

sns.heatmap((confusion_matrix(y_train_adb,y_train_pred_adb)),annot=True,fmt='.5g'
            ,cmap='Blues');
```

```
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

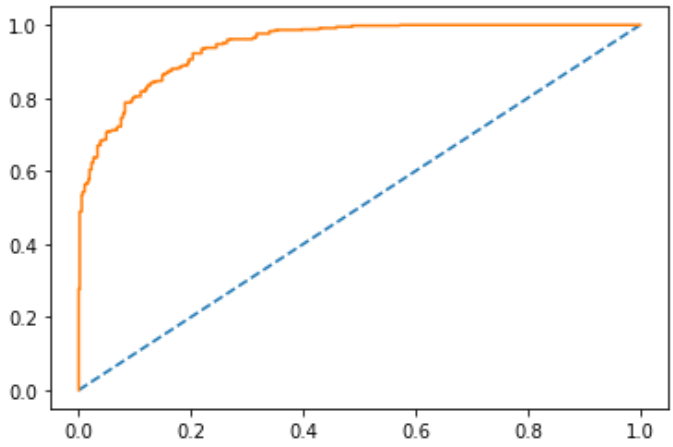
```
The classification report for Adaboosting training set is
              precision    recall  f1-score   support

         0.0       0.86      0.84      0.85       571
         1.0       0.85      0.87      0.86       593

    accuracy                           0.86      1164
   macro avg       0.86      0.86      0.86      1164
weighted avg       0.86      0.86      0.86      1164
```



In [103]:

```
probs_adb_train = abcl.predict_proba(X_train_adb)
probs_adb_train = probs_adb_train[:,1]
auc_train_adb = roc_auc_score(y_train_adb, probs_adb_train)
print('The AUC score for AdaBoosting training set is: %.3f'%auc_train_adb)

train_fpr_adb, train_tpr_adb, train_thresholds_adb = roc_curve(y_train_adb, probs_adb_tra
in);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_adb, train_tpr_adb);
```

```
The AUC score for AdaBoosting training set is: 0.945
```

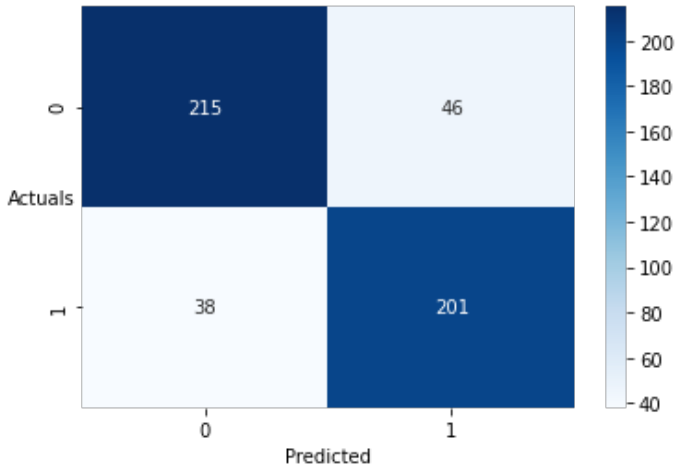


AdaBoosting on Test Set

In [104]:

```
print('The classification report for Adaboosting testing set is\n',classification_report(
y_test_adb, y_test_pred_adb))

sns.heatmap((confusion_matrix(y_test_adb,y_test_pred_adb)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

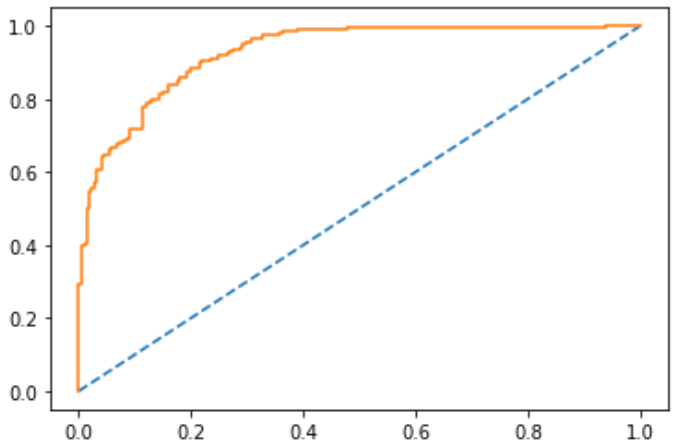
```
The classification report for Adaboosting testing set is
              precision    recall  f1-score   support

         0.0       0.85      0.82      0.84       261
         1.0       0.81      0.84      0.83       239

    accuracy                           0.83       500
   macro avg       0.83      0.83      0.83       500
weighted avg       0.83      0.83      0.83       500
```



In [105]:

```
probs_adb_test = abcl.predict_proba(X_test_adb)
probs_adb_test = probs_adb_test[:,1]
auc_test_adb = roc_auc_score(y_test_adb, probs_adb_test)
print('The AUC score for AdaBoosting testing set is: %.3f'%auc_test_adb)

test_fpr_adb, test_tpr_adb, test_thresholds_adb = roc_curve(y_test_adb, probs_adb_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_adb, test_tpr_adb);
```

```
The AUC score for AdaBoosting testing set is: 0.928
```



Gradient Boosting

In [106]:

```
from sklearn.ensemble import GradientBoostingClassifier

X_train_gdb, X_test_gdb, y_train_gdb, y_test_gdb = train_test_split(X_sm, y_sm, test_siz
e=0.3, random_state=42)

gbcl = GradientBoostingClassifier(n_estimators=50,random_state=1)
gbcl = gbcl.fit(X_train_gdb, y_train_gdb)
```

In [107]:

```
print('The model score for GradientBoosting training set is',gbcl.score(X_train_gdb,y_tra
in_gdb))
print('\n')
print('The model score for GradientBoosting testing set is',gbcl.score(X_test_gdb,y_test_
gdb))
```

```
The model score for GradientBoosting training set is 0.9390034364261168


The model score for GradientBoosting testing set is 0.91
```

In [108]:

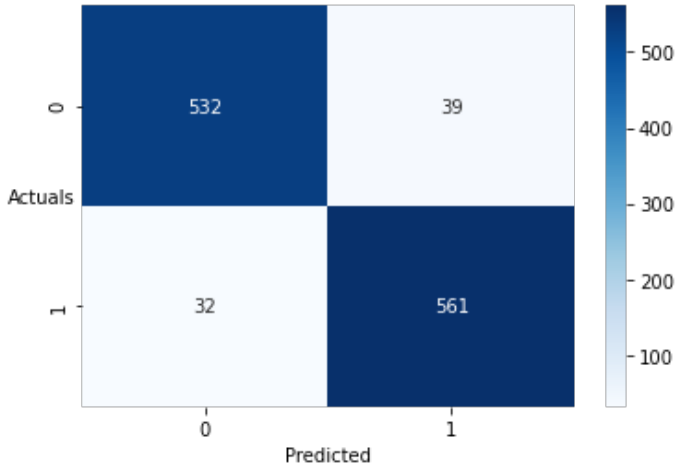
```
y_train_pred_gdb = gbcl.predict(X_train_gdb)
y_test_pred_gdb  = gbcl.predict(X_test_gdb)
```

Gradient Boosting on Train Set

In [109]:

```
print('The classification report for Gradientboosting training set is\n',classification_r
eport(y_train_gdb, y_train_pred_gdb))

sns.heatmap((confusion_matrix(y_train_gdb,y_train_pred_gdb)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

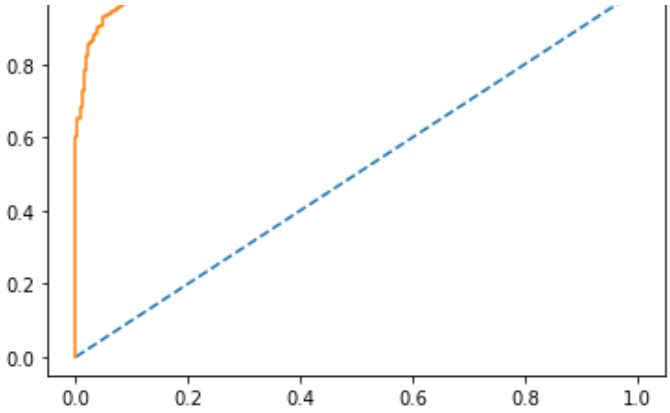
```
The classification report for Gradientboosting training set is
               precision    recall  f1-score   support

         0.0       0.94      0.93      0.94       571
         1.0       0.94      0.95      0.94       593

    accuracy                           0.94      1164
   macro avg       0.94      0.94      0.94      1164
weighted avg       0.94      0.94      0.94      1164
```



In [110]:

```
probs_gdb_train = gbcl.predict_proba(X_train_gdb)
probs_gdb_train = probs_gdb_train[:,1]
auc_train_gdb = roc_auc_score(y_train_gdb, probs_gdb_train)
print('The AUC score for GradientBoosting training set is: %.3f'%auc_train_gdb)

train_fpr_gdb, train_tpr_gdb, train_thresholds_gdb = roc_curve(y_train_gdb, probs_gdb_tra
in);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_gdb, train_tpr_gdb);
```

```
The AUC score for GradientBoosting training set is: 0.986
```

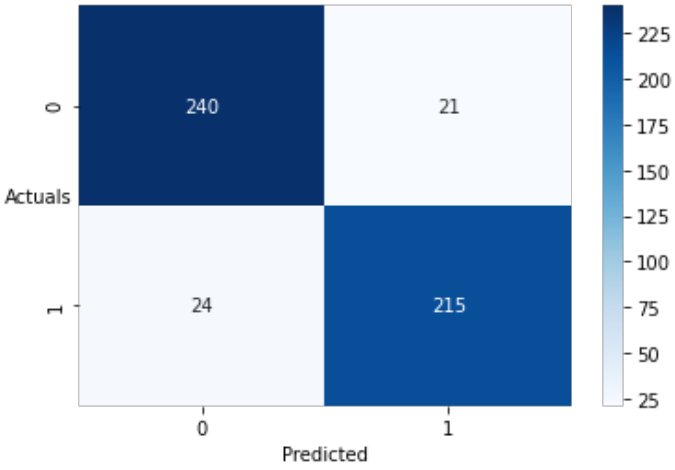


Gradient Boosting on Test Set

In [111]:

```
print('The classification report for Gradientboosting testing set is\n',classification_re
port(y_test_gdb, y_test_pred_gdb))

sns.heatmap((confusion_matrix(y_test_gdb,y_test_pred_gdb)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

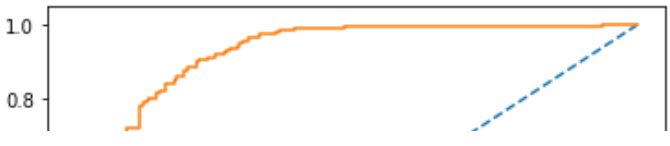
```
The classification report for Gradientboosting testing set is
              precision    recall  f1-score   support

         0.0       0.91      0.92      0.91       261
         1.0       0.91      0.90      0.91       239

    accuracy                           0.91       500
   macro avg       0.91      0.91      0.91       500
weighted avg       0.91      0.91      0.91       500
```

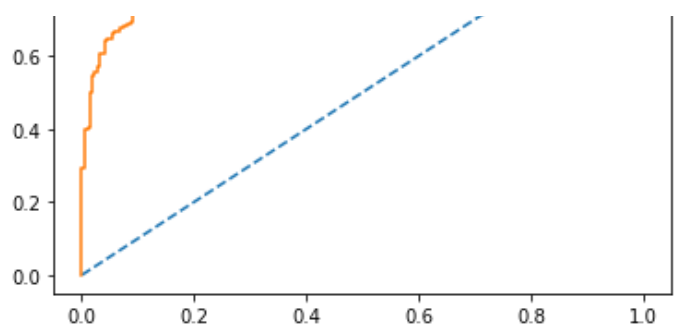


In [112]:

```
probs_gdb_test = abcl.predict_proba(X_test_gdb)
probs_gdb_test = probs_gdb_test[:,1]
auc_test_gdb = roc_auc_score(y_test_gdb, probs_gdb_test)
print('The AUC score for GradientBoosting testing set is: %.3f'%auc_test_gdb)

test_fpr_gdb, test_tpr_gdb, test_thresholds_gdb = roc_curve(y_test_gdb, probs_gdb_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_gdb, test_tpr_gdb);
```

```
The AUC score for GradientBoosting testing set is: 0.928
```





In [113]:

```
LR_Train_class = classification_report(y_train_LR, y_train_pred_LR , output_dict=True)
df_LR_Train    = pd.DataFrame(LR_Train_class).transpose()

LR_Train_precision = round(df_LR_Train.loc['1.0'][0] , 3)
LR_Train_recall    = round(df_LR_Train.loc['1.0'][1] , 3)
LR_Train_F1score   = round(df_LR_Train.loc['1.0'][2] , 3)
LR_Train_Accuracy  = round(df_LR_Train.loc['accuracy'][0], 3)
LR_Train_auc_score = round(auc_train , 3)
```

In [114]:

```
LR_Test_class = classification_report(y_test_LR, y_test_pred_LR , output_dict=True)
df_LR_Test    = pd.DataFrame(LR_Test_class).transpose()

LR_Test_precision = round(df_LR_Test.loc['1.0'][0] , 3)
LR_Test_recall    = round(df_LR_Test.loc['1.0'][1] , 3)
LR_Test_F1score   = round(df_LR_Test.loc['1.0'][2] , 3)
LR_Test_Accuracy  = round(df_LR_Test.loc['accuracy'][0], 3)
LR_Test_auc_score = round(auc_test , 3)
```

In [115]:

```
LDA_Train_class = classification_report(y_train_LDA, y_train_pred_LDA , output_dict=True
)
df_LDA_Train    = pd.DataFrame(LDA_Train_class).transpose()

LDA_Train_precision = round(df_LDA_Train.loc['1.0'][0] , 3)
LDA_Train_recall    = round(df_LDA_Train.loc['1.0'][1] , 3)
LDA_Train_F1score   = round(df_LDA_Train.loc['1.0'][2] , 3)
LDA_Train_Accuracy  = round(df_LDA_Train.loc['accuracy'][0], 3)
LDA_Train_auc_score = round(auc_train_LDA , 3)
```

In [116]:

```
LDA_Test_class = classification_report(y_test_LDA, y_test_pred_LDA , output_dict=True)
df_LDA_Test    = pd.DataFrame(LDA_Test_class).transpose()

LDA_Test_precision = round(df_LDA_Test.loc['1.0'][0] , 3)
LDA_Test_recall    = round(df_LDA_Test.loc['1.0'][1] , 3)
LDA_Test_F1score   = round(df_LDA_Test.loc['1.0'][2] , 3)
LDA_Test_Accuracy  = round(df_LDA_Test.loc['accuracy'][0], 3)
LDA_Test_auc_score = round(auc_test_LDA , 3)
```

In [117]:

```
KNN_Train_class = classification_report(y_train_KNN, y_train_pred_KNN , output_dict=True
)
df_KNN_Train    = pd.DataFrame(KNN_Train_class).transpose()

KNN_Train_precision = round(df_KNN_Train.loc['1.0'][0] , 3)
KNN_Train_recall    = round(df_KNN_Train.loc['1.0'][1] , 3)
KNN_Train_F1score   = round(df_KNN_Train.loc['1.0'][2] , 3)
KNN_Train_Accuracy  = round(df_KNN_Train.loc['accuracy'][0], 3)
KNN_Train_auc_score = round(auc_train_KNN , 3)
```

In [118]:

```
KNN_Test_class = classification_report(y_test_KNN, y_test_pred_KNN , output_dict=True)
```

```
KNN_Test_class = classification_report(y_test_KNN, y_test_pred_KNN , output_dict=True)
df_KNN_Test    = pd.DataFrame(KNN_Test_class).transpose()

KNN_Test_precision = round(df_KNN_Test.loc['1.0'][0] , 3)
KNN_Test_recall    = round(df_KNN_Test.loc['1.0'][1] , 3)
KNN_Test_F1score   = round(df_KNN_Test.loc['1.0'][2] , 3)
KNN_Test_Accuracy  = round(df_KNN_Test.loc['accuracy'][0], 3)
KNN_Test_auc_score = round(auc_test_KNN , 3)
```

In [119]:

```
NB_Train_class = classification_report(y_train_NB, y_train_pred_NB , output_dict=True)
df_NB_Train    = pd.DataFrame(NB_Train_class).transpose()

NB_Train_precision = round(df_NB_Train.loc['1.0'][0] , 3)
NB_Train_recall    = round(df_NB_Train.loc['1.0'][1] , 3)
NB_Train_F1score   = round(df_NB_Train.loc['1.0'][2] , 3)
NB_Train_Accuracy  = round(df_NB_Train.loc['accuracy'][0], 3)
NB_Train_auc_score = round(auc_train_NB , 3)
```

In [120]:

```
NB_Test_class = classification_report(y_test_NB, y_test_pred_NB , output_dict=True)
df_NB_Test    = pd.DataFrame(NB_Test_class).transpose()

NB_Test_precision = round(df_NB_Test.loc['1.0'][0] , 3)
NB_Test_recall    = round(df_NB_Test.loc['1.0'][1] , 3)
NB_Test_F1score   = round(df_NB_Test.loc['1.0'][2] , 3)
NB_Test_Accuracy  = round(df_NB_Test.loc['accuracy'][0], 3)
NB_Test_auc_score = round(auc_test_NB , 3)
```

In [121]:

```
DT_Train_class = classification_report(y_train_DT, y_train_pred_DT , output_dict=True)
df_DT_Train    = pd.DataFrame(DT_Train_class).transpose()

DT_Train_precision = round(df_DT_Train.loc['1.0'][0] , 3)
DT_Train_recall    = round(df_DT_Train.loc['1.0'][1] , 3)
DT_Train_F1score   = round(df_DT_Train.loc['1.0'][2] , 3)
DT_Train_Accuracy  = round(df_DT_Train.loc['accuracy'][0], 3)
DT_Train_auc_score = round(auc_train_DT, 3)
```

In [122]:

```
DT_Test_class = classification_report(y_test_DT, y_test_pred_DT , output_dict=True)
df_DT_Test    = pd.DataFrame(DT_Test_class).transpose()

DT_Test_precision = round(df_DT_Test.loc['1.0'][0] , 3)
DT_Test_recall    = round(df_DT_Test.loc['1.0'][1] , 3)
DT_Test_F1score   = round(df_DT_Test.loc['1.0'][2] , 3)
DT_Test_Accuracy  = round(df_DT_Test.loc['accuracy'][0], 3)
DT_Test_auc_score = round(auc_test_DT, 3)
```

In [123]:

```
RFC_Train_class = classification_report(y_train_RFC, y_train_pred_RFC , output_dict=True
)
df_RFC_Train    = pd.DataFrame(RFC_Train_class).transpose()

RFC_Train_precision = round(df_RFC_Train.loc['1.0'][0] , 3)
RFC_Train_recall    = round(df_RFC_Train.loc['1.0'][1] , 3)
RFC_Train_F1score   = round(df_RFC_Train.loc['1.0'][2] , 3)
RFC_Train_Accuracy  = round(df_RFC_Train.loc['accuracy'][0], 3)
RFC_Train_auc_score = round(auc_train_RFC, 3)
```

In [124]:

```
RFC_Test_class = classification_report(y_test_RFC, y_test_pred_RFC , output_dict=True)
df_RFC_Test    = pd.DataFrame(RFC_Test_class).transpose()

RFC_Test_precision = round(df_RFC_Test.loc['1.0'][0] , 3)
```

```
RFC_Test_recall    = round(df_RFC_Test.loc['1.0'][1] , 3)
RFC_Test_F1score   = round(df_RFC_Test.loc['1.0'][2] , 3)
RFC_Test_Accuracy  = round(df_RFC_Test.loc['accuracy'][0], 3)
RFC_Test_auc_score = round(auc_test_RFC, 3)
```

In [125]:

```
bg_Train_class = classification_report(y_train_bg, y_train_pred_bg , output_dict=True)
df_bg_Train    = pd.DataFrame(bg_Train_class).transpose()

bg_Train_precision = round(df_bg_Train.loc['1.0'][0] , 3)
bg_Train_recall    = round(df_bg_Train.loc['1.0'][1] , 3)
bg_Train_F1score   = round(df_bg_Train.loc['1.0'][2] , 3)
bg_Train_Accuracy  = round(df_bg_Train.loc['accuracy'][0], 3)
bg_Train_auc_score = round(auc_train_bg, 3)
```

In [126]:

```
bg_Test_class = classification_report(y_test_bg, y_test_pred_bg , output_dict=True)
df_bg_Test    = pd.DataFrame(bg_Test_class).transpose()

bg_Test_precision = round(df_bg_Test.loc['1.0'][0] , 3)
bg_Test_recall    = round(df_bg_Test.loc['1.0'][1] , 3)
bg_Test_F1score   = round(df_bg_Test.loc['1.0'][2] , 3)
bg_Test_Accuracy  = round(df_bg_Test.loc['accuracy'][0], 3)
bg_Test_auc_score = round(auc_test_bg, 3)
```

In [127]:

```
adb_Train_class = classification_report(y_train_adb, y_train_pred_adb , output_dict=True
)
df_adb_Train    = pd.DataFrame(adb_Train_class).transpose()

adb_Train_precision = round(df_adb_Train.loc['1.0'][0] , 3)
adb_Train_recall    = round(df_adb_Train.loc['1.0'][1] , 3)
adb_Train_F1score   = round(df_adb_Train.loc['1.0'][2] , 3)
adb_Train_Accuracy  = round(df_adb_Train.loc['accuracy'][0], 3)
adb_Train_auc_score = round(auc_train_adb, 3)
```

In [128]:

```
adb_Test_class = classification_report(y_test_adb, y_test_pred_adb , output_dict=True)
df_adb_Test    = pd.DataFrame(adb_Test_class).transpose()

adb_Test_precision = round(df_adb_Test.loc['1.0'][0] , 3)
adb_Test_recall    = round(df_adb_Test.loc['1.0'][1] , 3)
adb_Test_F1score   = round(df_adb_Test.loc['1.0'][2] , 3)
adb_Test_Accuracy  = round(df_adb_Test.loc['accuracy'][0], 3)
adb_Test_auc_score = round(auc_test_adb, 3)
```

In [129]:

```
gdb_Train_class = classification_report(y_train_gdb, y_train_pred_gdb , output_dict=True
)
df_gdb_Train    = pd.DataFrame(gdb_Train_class).transpose()

gdb_Train_precision = round(df_gdb_Train.loc['1.0'][0] , 3)
gdb_Train_recall    = round(df_gdb_Train.loc['1.0'][1] , 3)
gdb_Train_F1score   = round(df_gdb_Train.loc['1.0'][2] , 3)
gdb_Train_Accuracy  = round(df_gdb_Train.loc['accuracy'][0], 3)
gdb_Train_auc_score = round(auc_train_gdb, 3)
```

In [130]:

```
gdb_Test_class = classification_report(y_test_gdb, y_test_pred_gdb , output_dict=True)
df_gdb_Test    = pd.DataFrame(gdb_Test_class).transpose()

gdb_Test_precision = round(df_gdb_Test.loc['1.0'][0] , 3)
gdb_Test_recall    = round(df_gdb_Test.loc['1.0'][1] , 3)
gdb_Test_F1score   = round(df_gdb_Test.loc['1.0'][2] , 3)
```

```
gdb_Test_Accuracy  = round(df_gdb_Test.loc['accuracy'][0], 3)
gdb_Test_auc_score = round(auc_test_gdb, 3)
```

In [131]:

```
pd.set_option('display.max_columns', None)
index = ['Precision','Recall','F1 Score','Accuracy','AUC Score']
pd.DataFrame({'LR Train' :[LR_Train_precision , LR_Train_recall ,LR_Train_F1score, LR_Train_Accuracy, LR_Train_auc_score],
              'LR Test':[LR_Test_precision, LR_Test_recall, LR_Test_F1score, LR_Test_Accuracy,LR_Test_auc_score],
              'LDA Train'  :[LDA_Train_precision, LDA_Train_recall, LDA_Train_F1score, LDA_Train_Accuracy, LDA_Train_auc_score],
              'LDA Test' :[LDA_Test_precision,LDA_Test_recall,LDA_Test_F1score,LDA_Test_Accuracy,LDA_Test_auc_score],
              'KNN Train' :[KNN_Train_precision , KNN_Train_recall ,KNN_Train_F1score, KNN_Train_Accuracy, KNN_Train_auc_score],
              'KNN Test':[KNN_Test_precision, KNN_Test_recall, KNN_Test_F1score, KNN_Test_Accuracy,KNN_Test_auc_score],
              'NB Train'  :[NB_Train_precision, NB_Train_recall, NB_Train_F1score, NB_Train_Accuracy, NB_Train_auc_score],
              'NB Test' :[NB_Test_precision, NB_Test_recall, NB_Test_F1score, NB_Test_Accuracy, NB_Test_auc_score],
              'CART Train':[DT_Train_precision , DT_Train_recall ,DT_Train_F1score, DT_Train_Accuracy, DT_Train_auc_score],
              'CART Test':[DT_Test_precision, DT_Test_recall, DT_Test_F1score, DT_Test_Accuracy,DT_Test_auc_score],
              'RFC Train':[RFC_Train_precision , RFC_Train_recall ,RFC_Train_F1score, RFC_Train_Accuracy, RFC_Train_auc_score],
              'RFC Test':[RFC_Test_precision, RFC_Test_recall, RFC_Test_F1score, RFC_Test_Accuracy,RFC_Test_auc_score],
              'Bagging Train':[bg_Train_precision , bg_Train_recall ,bg_Train_F1score, bg_Train_Accuracy, bg_Train_auc_score],
              'Bagging Test':[bg_Test_precision, bg_Test_recall, bg_Test_F1score, bg_Test_Accuracy,bg_Test_auc_score],
              'Ada Boosting Train':[adb_Train_precision, adb_Train_recall, adb_Train_F1score, adb_Train_Accuracy, adb_Train_auc_score],
              'Ada Boosting Test':[adb_Test_precision,adb_Test_recall,adb_Test_F1score,adb_Test_Accuracy,adb_Test_auc_score],
              'Gradient Boosting Train':[gdb_Train_precision, gdb_Train_recall, gdb_Train_F1score, gdb_Train_Accuracy, gdb_Train_auc_score],
              'Gradient Boosting Test':[gdb_Test_precision,gdb_Test_recall,gdb_Test_F1score,gdb_Test_Accuracy,gdb_Test_auc_score],},index= index)
```

Out[131]:

	LR Train	LR Test	LDA Train	LDA Test	KNN Train	KNN Test	NB Train	NB Test	CART Train	CART Test	RFC Train	RFC Test	Bagging Train	Bagging Test	Ada Boosting Train	Boos
Precision	0.738	0.747	0.737	0.744	0.966	0.905	0.680	0.682	1.0	0.954	1.0	0.992	1.0	0.967	0.848	0
Recall	0.788	0.728	0.806	0.741	1.000	1.000	0.843	0.816	1.0	0.958	1.0	0.983	1.0	0.971	0.874	0
F1 Score	0.762	0.737	0.770	0.742	0.983	0.950	0.753	0.743	1.0	0.956	1.0	0.987	1.0	0.969	0.860	0
Accuracy	0.749	0.752	0.754	0.754	0.982	0.950	0.718	0.730	1.0	0.958	1.0	0.988	1.0	0.970	0.856	0
AUC Score	0.832	0.827	0.831	0.826	1.000	1.000	0.816	0.816	1.0	0.958	1.0	1.000	1.0	0.998	0.945	0

Mobiles

In [132]:

```
df4 = df2[df2['Laptop/Mobile_lbl']==1]
```

In [133]:

```
y = df4['Taken_product']
```

```
X = df4.drop('Taken_product',axis=1)
```

In [134]:

```
X_fit = sc.fit_transform(X)
X = pd.DataFrame(X_fit,columns=X.columns)
```

In [135]:

```
sm = SMOTE(random_state = 42)
X_sm, y_sm = sm.fit_resample(X, y)
```

In [136]:

```
X_train_LR, X_test_LR, y_train_LR, y_test_LR = train_test_split(X_sm, y_sm, test_size=0.3
, random_state=42)

model_LR = LogisticRegression()
model_LR.fit(X_train_LR, y_train_LR)
```

Out[136]:

```
LogisticRegression()
```

In [137]:

```
print('The model score for Logistic Regression training set is',model_LR.score(X_train_LR
, y_train_LR))
print('\n')
print('The model score for Logistic Regression testing set is',model_LR.score(X_test_LR,
y_test_LR))
```

```
The model score for Logistic Regression training set is 0.7230306864916166


The model score for Logistic Regression testing set is 0.7271217712177122
```

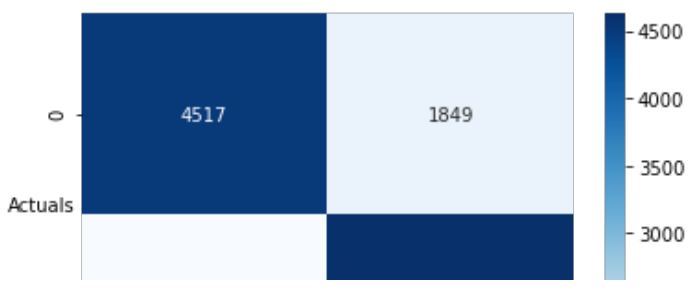
In [138]:

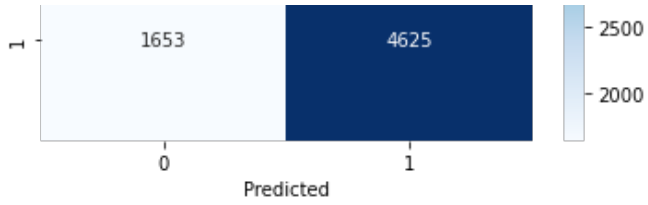
```
y_train_pred_LR = model_LR.predict(X_train_LR)
y_test_pred_LR  = model_LR.predict(X_test_LR)
```

In [139]:

```
sns.heatmap((confusion_matrix(y_train_LR,y_train_pred_LR)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);

print('The classification report for Logistic Regression training set is\n',classificatio
n_report(y_train_LR, y_train_pred_LR))
```

```
The classification report for Logistic Regression training set is
              precision    recall  f1-score   support

         0.0       0.73      0.71      0.72      6366
         1.0       0.71      0.74      0.73      6278

    accuracy                           0.72     12644
   macro avg       0.72      0.72      0.72     12644
weighted avg       0.72      0.72      0.72     12644
```

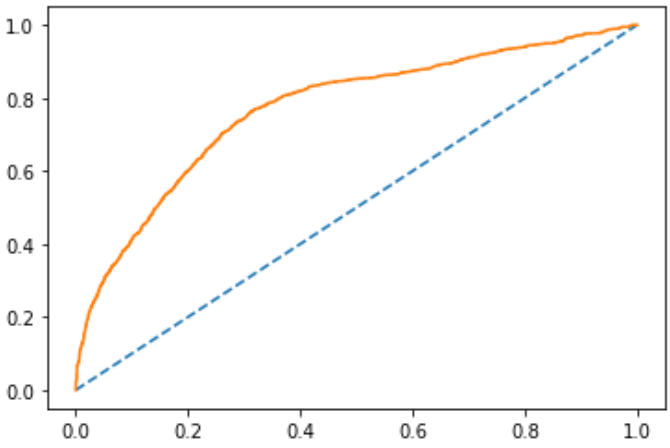




In [140]:

```
probs_LR_train = model_LR.predict_proba(X_train_LR)
probs_LR_train = probs_LR_train[:,1]
auc_train = roc_auc_score(y_train_LR, probs_LR_train)
print('The AUC score for Logistic Regression training set is: %.3f'%auc_train)

train_fpr_LR, train_tpr_LR, train_thresholds_LR = roc_curve(y_train_LR, probs_LR_train);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_LR, train_tpr_LR);
```

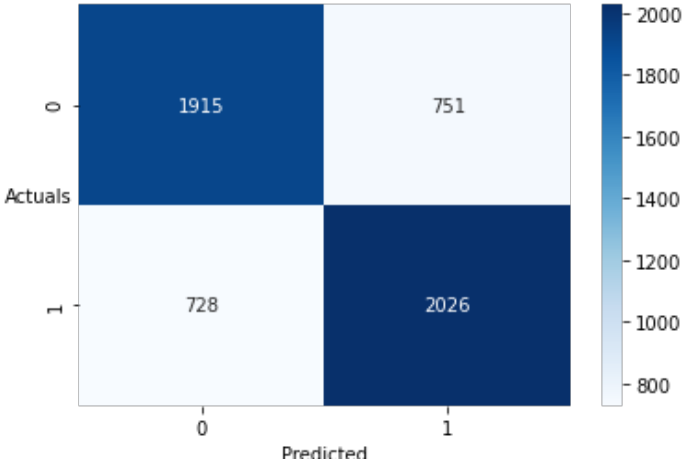
```
The AUC score for Logistic Regression training set is: 0.771
```



In [141]:

```
print('The classification report for Logistic Regression testing set is\n',classification
_report(y_test_LR, y_test_pred_LR))

sns.heatmap((confusion_matrix(y_test_LR,y_test_pred_LR)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

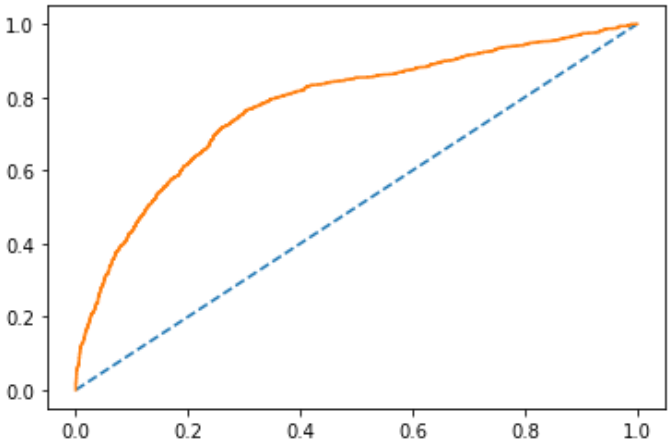
```
The classification report for Logistic Regression testing set is
              precision    recall  f1-score   support

         0.0       0.72      0.72      0.72      2666
         1.0       0.73      0.74      0.73      2754

    accuracy                           0.73      5420
   macro avg       0.73      0.73      0.73      5420
weighted avg       0.73      0.73      0.73      5420
```



In [142]:

```
probs_LR_test = model_LR.predict_proba(X_test_LR)
probs_LR_test = probs_LR_test[:,1]
auc_test = roc_auc_score(y_test_LR, probs_LR_test)
print('The AUC score for Logistic Regression testing set is: %.3f'%auc_test)

test_fpr_LR, test_tpr_LR, test_thresholds_LR = roc_curve(y_test_LR, probs_LR_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_LR, test_tpr_LR);
```

```
The AUC score for Logistic Regression testing set is: 0.777
```



Linear Discriminant Analysis

In [143]:

```
X_train_LDA, X_test_LDA, y_train_LDA, y_test_LDA = train_test_split(X_sm, y_sm, test_siz
e=0.3, random_state=42)

model_LDA = LinearDiscriminantAnalysis()
model_LDA.fit(X_train_LDA, y_train_LDA)
```

Out[143]:

```
LinearDiscriminantAnalysis()
```

In [144]:

```
print('The model score for Linear Discriminant Analysis training set is',model_LDA.score(
X_train_LDA, y_train_LDA))
print('\n')
print('The model score for Linear Discriminant Analysis testing set is',model_LDA.score(X
_test_LDA, y_test_LDA))
```

```
The model score for Linear Discriminant Analysis training set is 0.7220816197405884


The model score for Linear Discriminant Analysis testing set is 0.7271217712177122
```

In [145]:

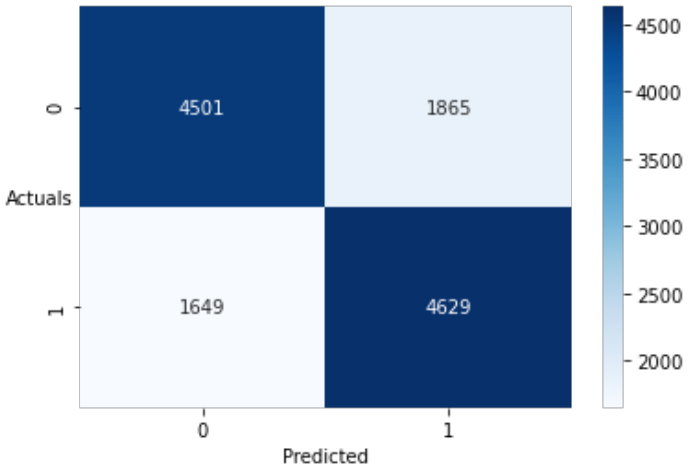
```
y_train_pred_LDA = model_LDA.predict(X_train_LDA)
y_test_pred_LDA  = model_LDA.predict(X_test_LDA)
```

In [146]:

```
print('The classification report for Linear Discriminant Analysis training set is\n',clas
sification_report(y_train_LDA, y_train_pred_LDA))

sns.heatmap((confusion_matrix(y_train_LDA,y_train_pred_LDA)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
```

```
plt.ylabel('Actuals',rotation=0);
```

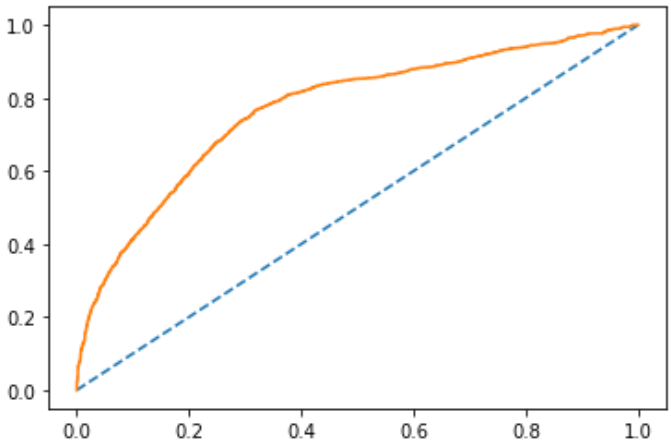
```
The classification report for Linear Discriminant Analysis training set is
              precision    recall  f1-score   support

         0.0       0.73      0.71      0.72      6366
         1.0       0.71      0.74      0.72      6278

    accuracy                           0.72     12644
   macro avg       0.72      0.72      0.72     12644
weighted avg       0.72      0.72      0.72     12644
```



In [147]:

```
probs_LDA_train = model_LDA.predict_proba(X_train_LDA)
probs_LDA_train = probs_LDA_train[:,1]
auc_train_LDA = roc_auc_score(y_train_LDA, probs_LDA_train)
print('The AUC score for Linear Discriminant Analysis training set is: %.3f'%auc_train_LD
A)

train_fpr_LDA, train_tpr_LDA, train_thresholds_LDA = roc_curve(y_train_LDA, probs_LDA_tra
in);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_LDA, train_tpr_LDA);
```

```
The AUC score for Linear Discriminant Analysis training set is: 0.770
```

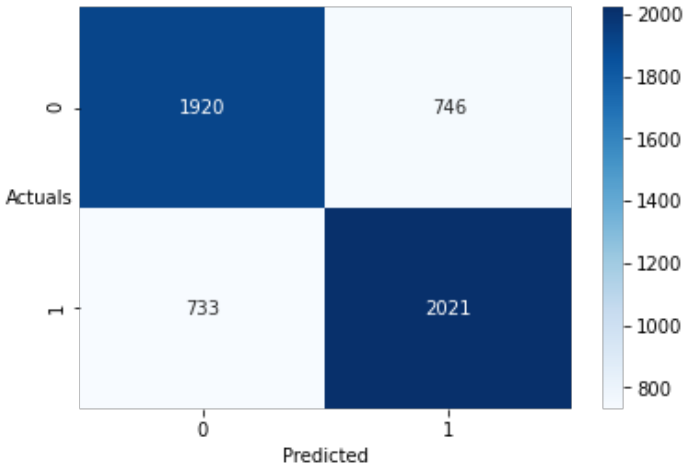


In [216]:

```
print('The classification report for Linear Discriminant Analysis testing set is\n',class
ification_report(y_test_LDA, y_test_pred_LDA))

sns.heatmap((confusion_matrix(y_test_LDA,y_test_pred_LDA)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

```
The classification report for Linear Discriminant Analysis testing set is
              precision    recall  f1-score   support
```

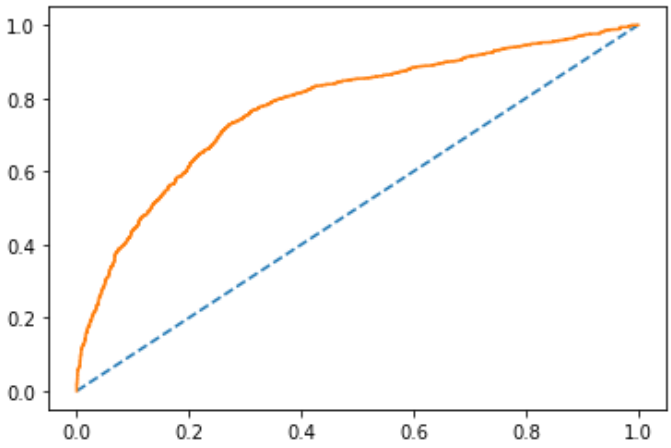
```
         0.0       0.72      0.72      0.72      2666
         1.0       0.73      0.73      0.73      2754

    accuracy                           0.73      5420
   macro avg       0.73      0.73      0.73      5420
weighted avg       0.73      0.73      0.73      5420
```



In [222]:

```
probs_LDA_test = model_LDA.predict_proba(X_test_LDA)
probs_LDA_test = probs_LDA_test[:,1]
auc_test_LDA = roc_auc_score(y_test_LDA, probs_LDA_test)
print('The AUC score for Linear Discriminant Analysis testing set is: %.3f'%auc_test_LDA)

test_fpr_LDA, test_tpr_LDA, test_thresholds_LDA = roc_curve(y_test_LDA, probs_LDA_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_LDA, test_tpr_LDA);
```

```
The AUC score for Linear Discriminant Analysis testing set is: 0.776
```



K- Nearest Neighbours

In [148]:

```
X_train_KNN, X_test_KNN, y_train_KNN, y_test_KNN = train_test_split(X_sm, y_sm, test_siz
e=0.30, random_state=42)

model_KNN = KNeighborsClassifier()
model_KNN.fit(X_train_KNN, y_train_KNN)
```

Out[148]:

```
KNeighborsClassifier()
```

In [149]:

```
print('The model score for KNN training set is',model_KNN.score(X_train_KNN, y_train_KNN)
```

```
)
print('\n')
print('The model score for KNN testing set is',model_KNN.score(X_test_KNN, y_test_KNN))
```

```
The model score for KNN training set is 0.9929610882632078


The model score for KNN testing set is 0.9859778597785978
```

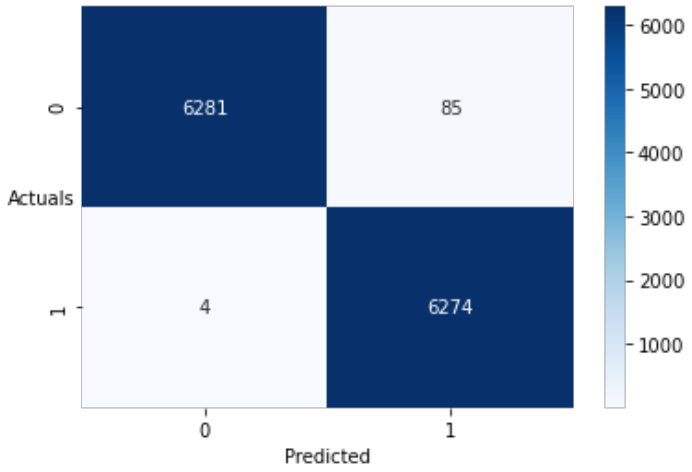
In [150]:

```
y_train_pred_KNN = model_KNN.predict(X_train_KNN)
y_test_pred_KNN  = model_KNN.predict(X_test_KNN)
```

In [151]:

```
print('The classification report for KNN set is\n',classification_report(y_train_KNN, y_t
rain_pred_KNN))

sns.heatmap((confusion_matrix(y_train_KNN,y_train_pred_KNN)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

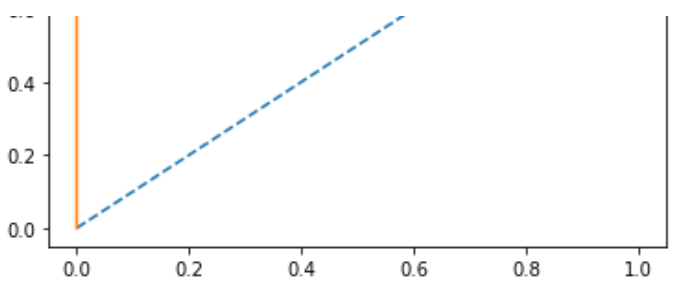
```
The classification report for KNN set is
              precision    recall  f1-score   support

         0.0       1.00      0.99      0.99      6366
         1.0       0.99      1.00      0.99      6278

    accuracy                           0.99     12644
   macro avg       0.99      0.99      0.99     12644
weighted avg       0.99      0.99      0.99     12644
```



In [152]:

```
probs_KNN_train = model_KNN.predict_proba(X_train_KNN)
probs_KNN_train = probs_KNN_train[:,1]
auc_train_KNN = roc_auc_score(y_train_KNN, probs_KNN_train)

print('The AUC score for KNN training set is: %.3f'%auc_train_KNN)

train_fpr_KNN, train_tpr_KNN, train_thresholds_KNN = roc_curve(y_train_KNN, probs_KNN_tra
in);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_KNN, train_tpr_KNN);
```

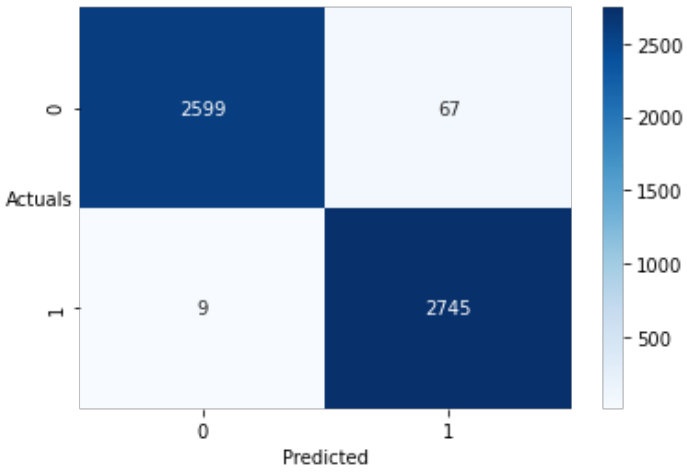
```
The AUC score for KNN training set is: 1.000
```



In [153]:

```
print('The classification report for KNN testing set is\n',classification_report(y_test_KNN, y_test_pred_KNN))

sns.heatmap((confusion_matrix(y_test_KNN,y_test_pred_KNN)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

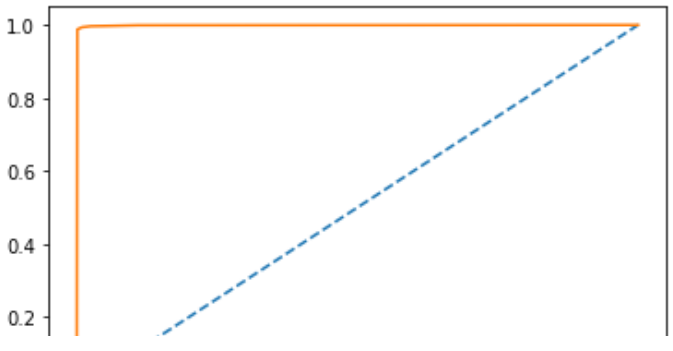
```
The classification report for KNN testing set is
              precision    recall  f1-score   support

         0.0       1.00      0.97      0.99      2666
         1.0       0.98      1.00      0.99      2754

    accuracy                           0.99      5420
   macro avg       0.99      0.99      0.99      5420
weighted avg       0.99      0.99      0.99      5420
```

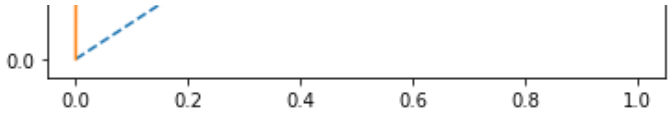


In [154]:

```
probs_KNN_test = model_KNN.predict_proba(X_test_KNN)
probs_KNN_test = probs_KNN_test[:,1]
auc_test_KNN = roc_auc_score(y_test_KNN, probs_KNN_test)

print('The AUC score for KNN testing set is: %.3f'%auc_test_KNN)

test_fpr_KNN, test_tpr_KNN, test_thresholds_KNN = roc_curve(y_test_KNN, probs_KNN_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_KNN, test_tpr_KNN);
```

```
The AUC score for KNN testing set is: 0.999
```





Naive Bayes

In [155]:

```
X_train_NB, X_test_NB, y_train_NB, y_test_NB = train_test_split(X_sm, y_sm, test_size=0.3
0, random_state=42)

model_NB = GaussianNB()
model_NB.fit(X_train_NB, y_train_NB)
```

Out[155]:

```
GaussianNB()
```

In [156]:

```
print('The model score for Naive Bayes Model training set is',model_NB.score(X_train_NB,
y_train_NB))
print('\n')
print('The model score for Naive Bayes Model testing set is',model_NB.score(X_test_NB, y_
test_NB))
```

```
The model score for Naive Bayes Model training set is 0.6907624169566593


The model score for Naive Bayes Model testing set is 0.6863468634686347
```

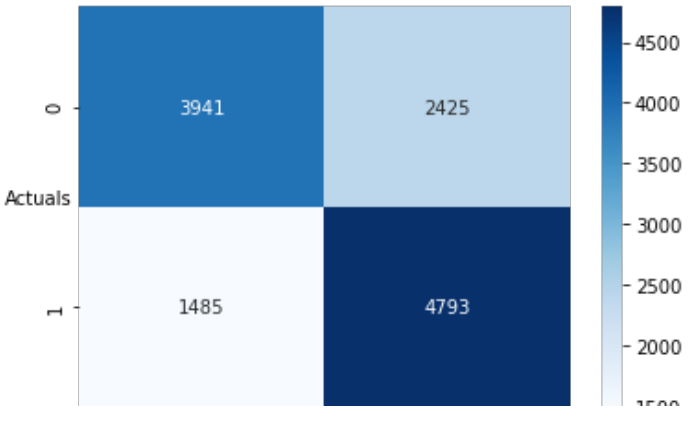
In [157]:

```
y_train_pred_NB = model_NB.predict(X_train_NB)
y_test_pred_NB  = model_NB.predict(X_test_NB)
```

In [158]:

```
print('The classification report for Naive Bayes Model set is\n',classification_report(y_
train_NB, y_train_pred_NB))

sns.heatmap((confusion_matrix(y_train_NB,y_train_pred_NB)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

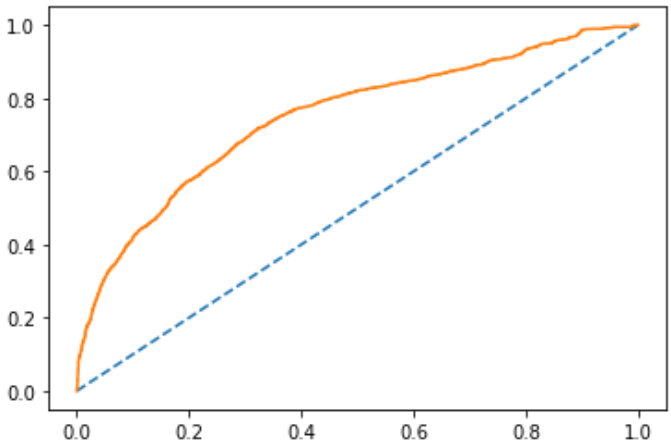
```
The classification report for Naive Bayes Model set is
               precision    recall  f1-score   support

         0.0       0.73      0.62      0.67      6366
         1.0       0.66      0.76      0.71      6278

    accuracy                           0.69     12644
   macro avg       0.70      0.69      0.69     12644
weighted avg       0.70      0.69      0.69     12644
```



In [159]:

```
probs_NB_train = model_NB.predict_proba(X_train_NB)
probs_NB_train = probs_NB_train[:,1]
auc_train_NB = roc_auc_score(y_train_NB, probs_NB_train)

print('The AUC score for Naive Bayes training set is: %.3f'%auc_train_NB)

train_fpr_NB, train_tpr_NB, train_thresholds_NB = roc_curve(y_train_NB, probs_NB_train);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_NB, train_tpr_NB);
```

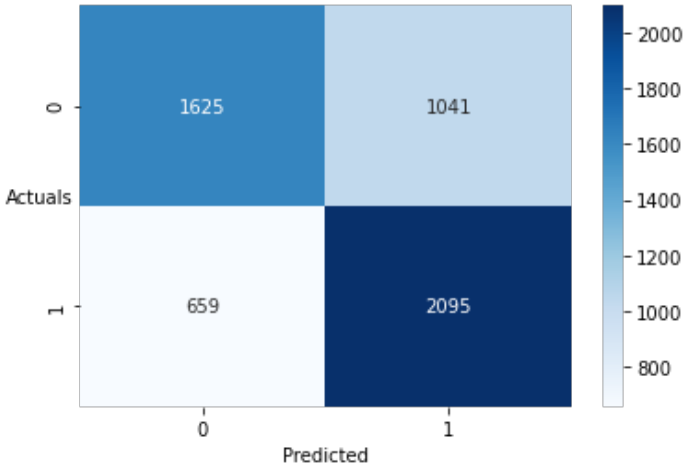
```
The AUC score for Naive Bayes training set is: 0.749
```



In [160]:

```
print('The classification report for Naive bayes Model testing set is\n',classification_r
eport(y_test_NB, y_test_pred_NB))

sns.heatmap((confusion_matrix(y_test_NB,y_test_pred_NB)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

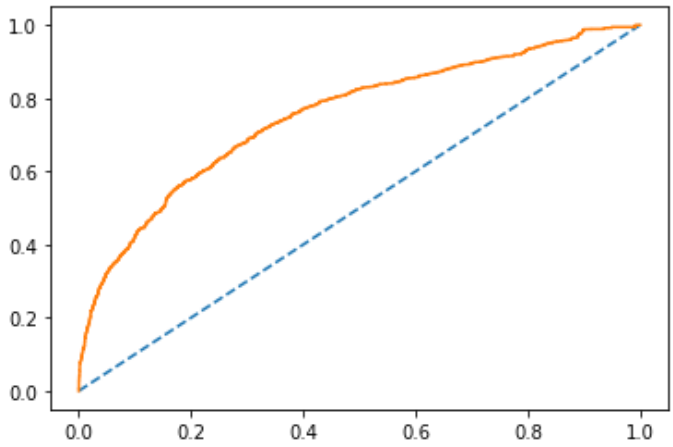
```
The classification report for Naive bayes Model testing set is
              precision    recall  f1-score   support

         0.0       0.71      0.61      0.66      2666
         1.0       0.67      0.76      0.71      2754

    accuracy                           0.69      5420
   macro avg       0.69      0.69      0.68      5420
weighted avg       0.69      0.69      0.68      5420
```



In [161]:

```
probs_NB_test = model_NB.predict_proba(X_test_NB)
probs_NB_test = probs_NB_test[:,1]
auc_test_NB   = roc_auc_score(y_test_NB, probs_NB_test)

print('The AUC score for Naive Bayes testing set is: %.3f'%auc_test_NB)

test_fpr_NB, test_tpr_NB, test_thresholds_NB = roc_curve(y_test_NB, probs_NB_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_NB, test_tpr_NB);
```

```
The AUC score for Naive Bayes testing set is: 0.753
```



Decision Tree Classifier

In [162]:

```
X_train_DT, X_test_DT, y_train_DT, y_test_DT = train_test_split(X_sm, y_sm, test_size=0.3
, random_state=42)

model_DT = DecisionTreeClassifier()
model_DT.fit(X_train_DT, y_train_DT)
```

Out[162]:

```
DecisionTreeClassifier()
```

In [163]:

```
print('The model score for Decision Tree Classifier training set is',model_DT.score(X_tra
in_DT,y_train_DT))
print('\n')
print('The model score for Decision Tree Classifier testing set is',model_DT.score(X_test
_DT,y_test_DT))
```

```
The model score for Decision Tree Classifier training set is 1.0


The model score for Decision Tree Classifier testing set is 0.9809963099630996
```

In [164]:

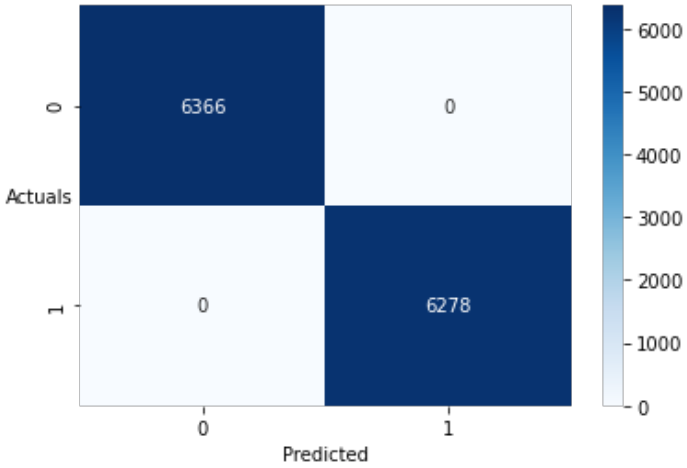
```
y_train_pred_DT = model_DT.predict(X_train_DT)
y_test_pred_DT  = model_DT.predict(X_test_DT)
```

In [165]:

```
print('The classification report for Decision Tree training set is\n',classification_repo
rt(y_train_DT, y_train_pred_DT))

sns.heatmap((confusion_matrix(y_train_DT,y_train_pred_DT)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

```
The classification report for Decision Tree training set is
              precision    recall  f1-score   support
```

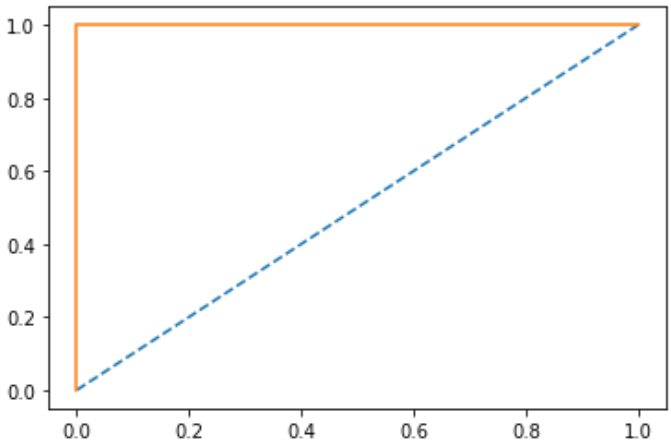
```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      6366
         1.0       1.00      1.00      1.00      6278

    accuracy                           1.00     12644
   macro avg       1.00      1.00      1.00     12644
weighted avg       1.00      1.00      1.00     12644
```



In [166]:

```
probs_DT_train = model_DT.predict_proba(X_train_DT)
probs_DT_train = probs_DT_train[:,1]
auc_train_DT = roc_auc_score(y_train_DT, probs_DT_train)
print('The AUC score for Decision Tree training set is: %.3f'%auc_train_DT)

train_fpr_DT, train_tpr_DT, train_thresholds_DT = roc_curve(y_train_DT, probs_DT_train);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_DT, train_tpr_DT);
```

```
The AUC score for Decision Tree training set is: 1.000
```

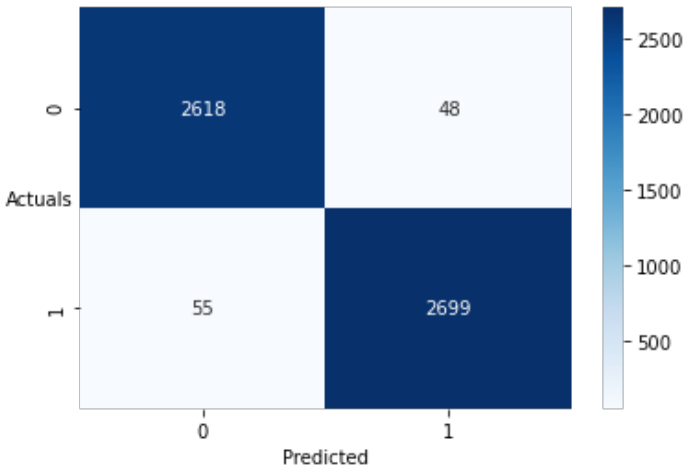


In [167]:

```
print('The classification report for Decision Tree testing set is\n',classification_repor
t(y_test_DT, y_test_pred_DT))

sns.heatmap((confusion_matrix(y_test_DT,y_test_pred_DT)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

```
The classification report for Decision Tree testing set is
              precision    recall  f1-score   support

         0.0       0.98      0.98      0.98      2666
         1.0       0.98      0.98      0.98      2754

    accuracy                           0.98      5420
   macro avg       0.98      0.98      0.98      5420
```

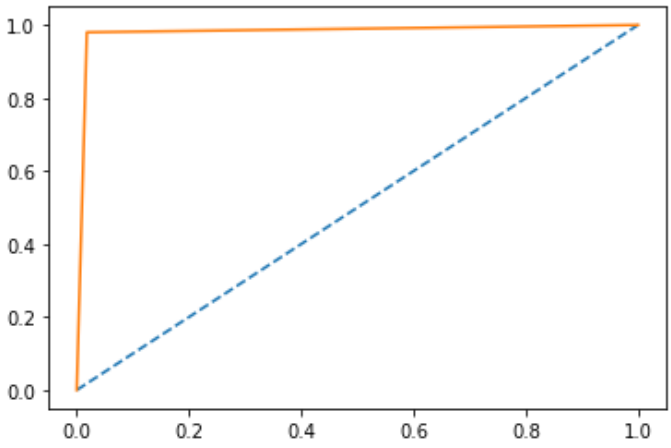
```
weighted avg       0.98      0.98      0.98      5420
```



In [168]:

```
probs_DT_test = model_DT.predict_proba(X_test_DT)
probs_DT_test = probs_DT_test[:,1]
auc_test_DT = roc_auc_score(y_test_DT, probs_DT_test)
print('The AUC score for Decision Tree testing set is: %.3f'%auc_test_DT)

test_fpr_DT, test_tpr_DT, test_thresholds_DT = roc_curve(y_test_DT, probs_DT_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_DT, test_tpr_DT);
```

```
The AUC score for Decision Tree testing set is: 0.981
```



Random Forest Classifier

In [169]:

```
X_train_RFC, X_test_RFC, y_train_RFC, y_test_RFC = train_test_split(X_sm, y_sm, test_siz
e=0.3, random_state=42)

rfcl = RandomForestClassifier()
rfcl = rfcl.fit(X_train_RFC, y_train_RFC)
```

In [170]:

```
print('The model score for Random Forest Classifier training set is',rfcl.score(X_train_R
FC,y_train_RFC))
print('\n')
print('The model score for Random Forest Classifier testing set is',rfcl.score(X_test_RFC
,y_test_RFC))
```

```
The model score for Random Forest Classifier training set is 1.0


The model score for Random Forest Classifier testing set is 0.9948339483394834
```

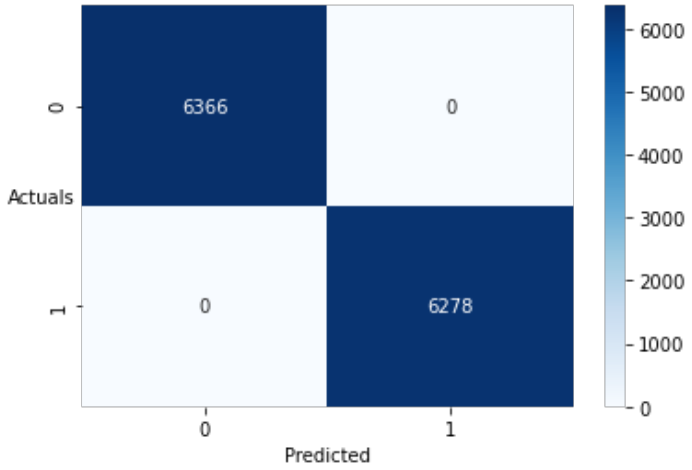
In [171]:

```
y_train_pred_RFC = rfcl.predict(X_train_RFC)
y_test_pred_RFC  = rfcl.predict(X_test_RFC)
```

In [172]:

```
print('The classification report for RFC training set is\n',classification_report(y_train
_RFC, y_train_pred_RFC))

sns.heatmap((confusion_matrix(y_train_RFC,y_train_pred_RFC)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

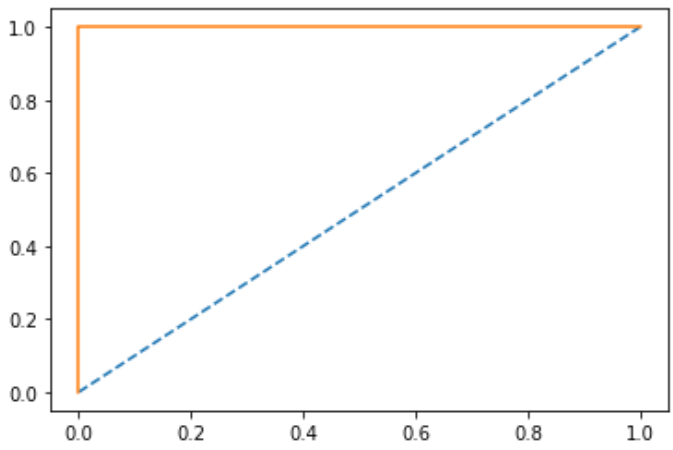
```
The classification report for RFC training set is
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      6366
         1.0       1.00      1.00      1.00      6278

    accuracy                           1.00     12644
   macro avg       1.00      1.00      1.00     12644
weighted avg       1.00      1.00      1.00     12644
```



In [173]:

```
probs_RFC_train = rfcl.predict_proba(X_train_RFC)
probs_RFC_train = probs_RFC_train[:,1]
auc_train_RFC = roc_auc_score(y_train_RFC, probs_RFC_train)
print('The AUC score for RFC training set is: %.3f'%auc_train_RFC)

train_fpr_RFC, train_tpr_RFC, train_thresholds_RFC = roc_curve(y_train_RFC, probs_RFC_tra
in);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_RFC, train_tpr_RFC);
```

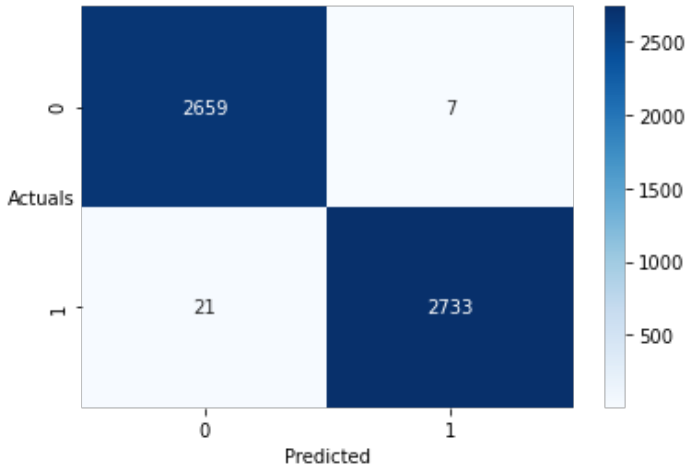
```
The AUC score for RFC training set is: 1.000
```



In [174]:

```
print('The classification report for RFC testing set is\n',classification_report(y_test_R
FC, y_test_pred_RFC))

sns.heatmap((confusion_matrix(y_test_RFC,y_test_pred_RFC)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

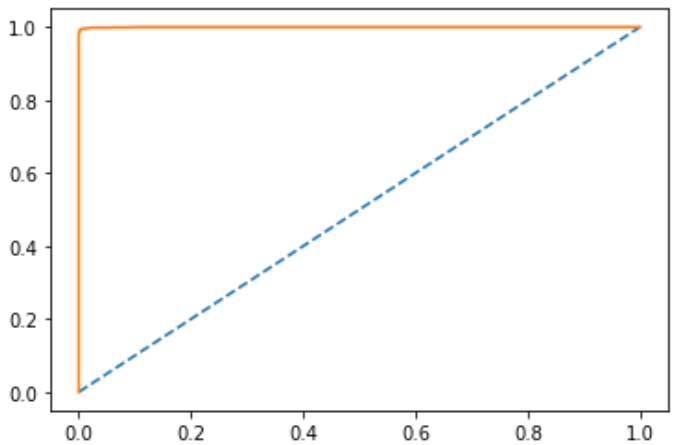
```
The classification report for RFC testing set is
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99      2666
         1.0       1.00      0.99      0.99      2754

    accuracy                           0.99      5420
   macro avg       0.99      0.99      0.99      5420
weighted avg       0.99      0.99      0.99      5420
```



In [175]:

```
probs_RFC_test = rfcl.predict_proba(X_test_RFC)
probs_RFC_test = probs_RFC_test[:,1]
auc_test_RFC = roc_auc_score(y_test_RFC, probs_RFC_test)
print('The AUC score for RFC testing set is: %.3f'%auc_test_RFC)

test_fpr_RFC, test_tpr_RFC, test_thresholds_RFC = roc_curve(y_test_RFC, probs_RFC_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_RFC, test_tpr_RFC);
```

```
The AUC score for RFC testing set is: 1.000
```



Model Tuning

Bagging

Bagging

In [176]:

```
X_train_bg, X_test_bg, y_train_bg, y_test_bg = train_test_split(X_sm, y_sm, test_size=0.3
, random_state=42)

bgcl = BaggingClassifier(n_estimators=50,random_state=1)
bgcl = bgcl.fit(X_train_bg, y_train_bg)
```

In [177]:

```
print('The model score for Bagging training set is',bgcl.score(X_train_bg,y_train_bg))
print('\n')
print('The model score for Bagging testing set is',bgcl.score(X_test_bg,y_test_bg))
```

```
The model score for Bagging training set is 1.0


The model score for Bagging testing set is 0.9920664206642067
```

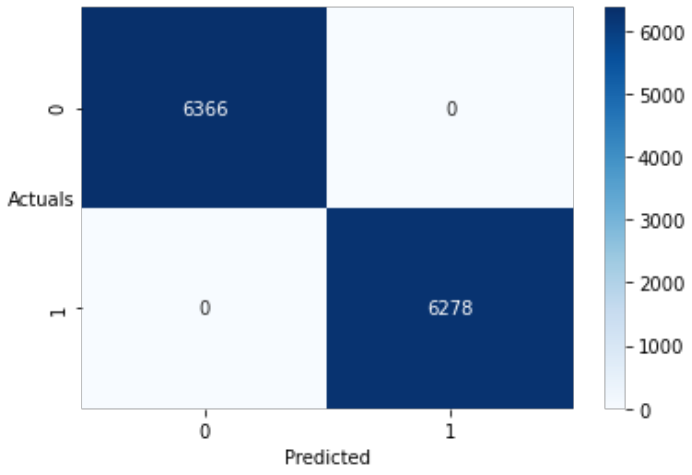
In [178]:

```
y_train_pred_bg = bgcl.predict(X_train_bg)
y_test_pred_bg  = bgcl.predict(X_test_bg)
```

In [179]:

```
print('The classification report for Bagging training set is\n',classification_report(y_t
rain_bg, y_train_pred_bg))

sns.heatmap((confusion_matrix(y_train_bg,y_train_pred_bg)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

```
The classification report for Bagging training set is
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      6366
         1.0       1.00      1.00      1.00      6278

    accuracy                           1.00     12644
   macro avg       1.00      1.00      1.00     12644
weighted avg       1.00      1.00      1.00     12644
```

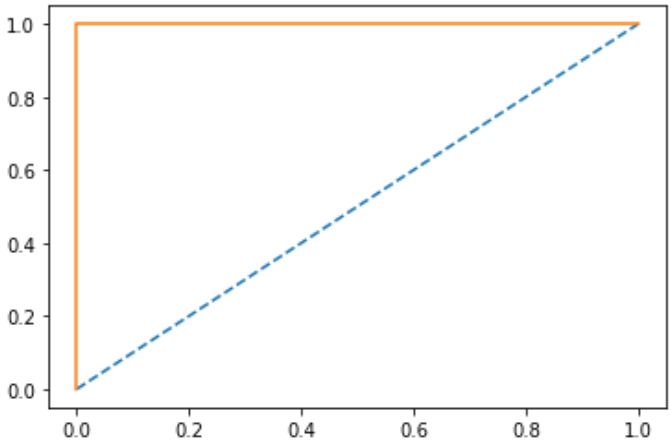


In [180]:

```
probs_bg_train = bgcl.predict_proba(X_train_bg)
probs_bg_train = probs_bg_train[:,1]
auc_train_bg = roc_auc_score(y_train_bg, probs_bg_train)
print('The AUC score for Bagging training set is: %.3f'%auc_train_bg)

train_fpr_bg, train_tpr_bg, train_thresholds_bg = roc_curve(y_train_bg, probs_bg_train);
```

```
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_bg, train_tpr_bg);
```

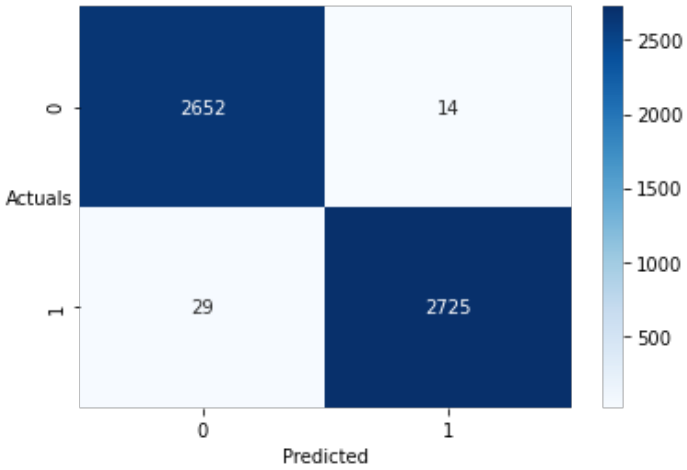
```
The AUC score for Bagging training set is: 1.000
```



In [181]:

```
print('The classification report for Bagging testing set is\n',classification_report(y_te
st_bg, y_test_pred_bg))

sns.heatmap((confusion_matrix(y_test_bg,y_test_pred_bg)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

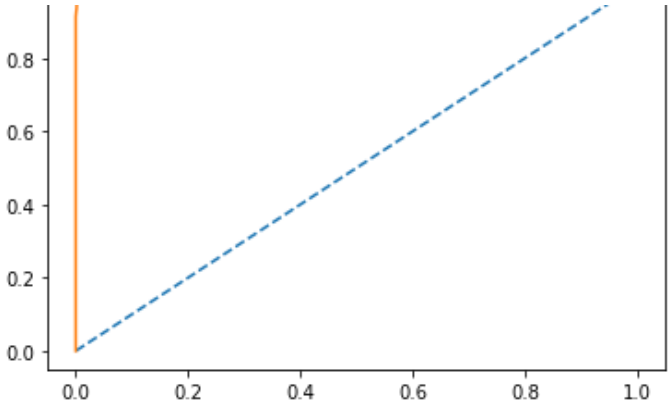
```
The classification report for Bagging testing set is
              precision    recall  f1-score   support

         0.0       0.99      0.99      0.99      2666
         1.0       0.99      0.99      0.99      2754

    accuracy                           0.99      5420
   macro avg       0.99      0.99      0.99      5420
weighted avg       0.99      0.99      0.99      5420
```



In [182]:

```
probs_bg_test = bgcl.predict_proba(X_test_bg)
probs_bg_test = probs_bg_test[:,1]
auc_test_bg = roc_auc_score(y_test_bg, probs_bg_test)
print('The AUC score for Bagging testing set is: %.3f'%auc_test_bg)

test_fpr_bg, test_tpr_bg, test_thresholds_bg = roc_curve(y_test_bg, probs_bg_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_bg, test_tpr_bg);
```

```
The AUC score for Bagging testing set is: 0.999
```



AdaBoosting

In [183]:

```
X_train_adb, X_test_adb, y_train_adb, y_test_adb = train_test_split(X_sm, y_sm, test_siz
e=0.3, random_state=42)

abcl = AdaBoostClassifier(n_estimators=50,random_state=1)
abcl = abcl.fit(X_train_adb, y_train_adb)
```

In [184]:

```
print('The model score for AdaBoosting training set is',abcl.score(X_train_adb,y_train_ad
b))
print('\n')
print('The model score for AdaBoosting testing set is',abcl.score(X_test_adb,y_test_adb))
```

```
The model score for AdaBoosting training set is 0.7797374248655489


The model score for AdaBoosting testing set is 0.7765682656826568
```

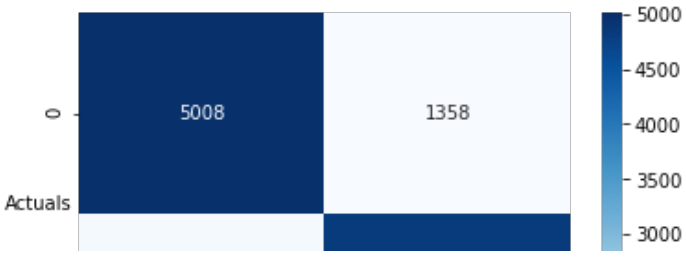
In [185]:

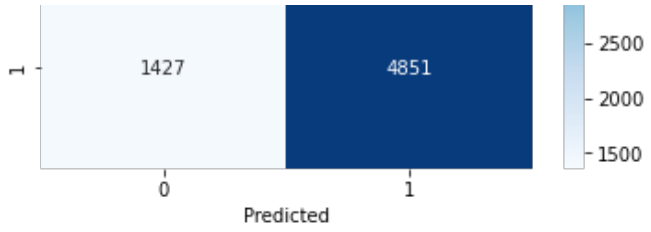
```
y_train_pred_adb = abcl.predict(X_train_adb)
y_test_pred_adb  = abcl.predict(X_test_adb)
```

In [186]:

```
print('The classification report for Adaboosting training set is\n',classification_report
(y_train_adb, y_train_pred_adb))

sns.heatmap((confusion_matrix(y_train_adb,y_train_pred_adb)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

```
The classification report for Adaboosting training set is
              precision    recall  f1-score   support

         0.0       0.78      0.79      0.78      6366
         1.0       0.78      0.77      0.78      6278

    accuracy                           0.78     12644
   macro avg       0.78      0.78      0.78     12644
weighted avg       0.78      0.78      0.78     12644
```

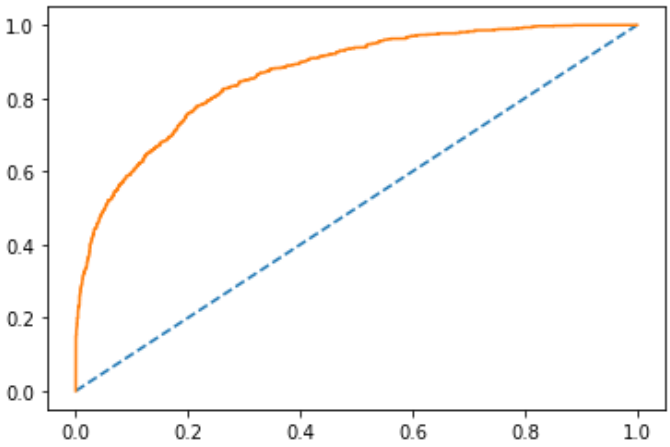




In [187]:

```
probs_adb_train = abcl.predict_proba(X_train_adb)
probs_adb_train = probs_adb_train[:,1]
auc_train_adb = roc_auc_score(y_train_adb, probs_adb_train)
print('The AUC score for AdaBoosting training set is: %.3f'%auc_train_adb)

train_fpr_adb, train_tpr_adb, train_thresholds_adb = roc_curve(y_train_adb, probs_adb_tra
in);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_adb, train_tpr_adb);
```

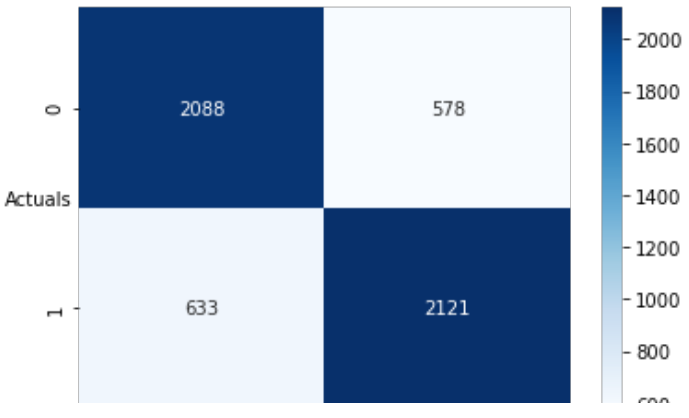
```
The AUC score for AdaBoosting training set is: 0.865
```



In [188]:

```
print('The classification report for Adaboosting testing set is\n',classification_report(
y_test_adb, y_test_pred_adb))

sns.heatmap((confusion_matrix(y_test_adb,y_test_pred_adb)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

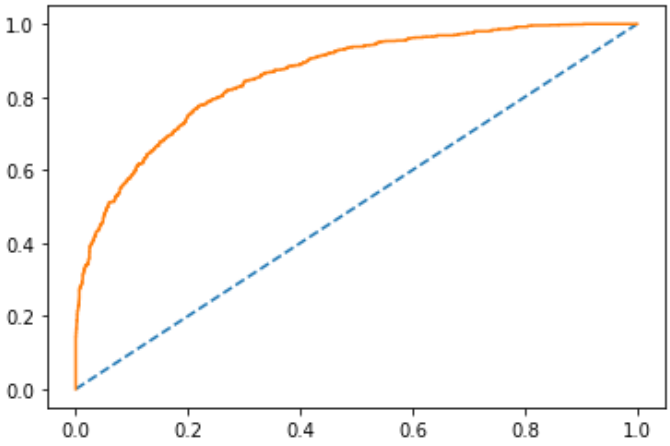
```
The classification report for Adaboosting testing set is
              precision    recall  f1-score   support

         0.0       0.77      0.78      0.78      2666
         1.0       0.79      0.77      0.78      2754

    accuracy                           0.78      5420
   macro avg       0.78      0.78      0.78      5420
weighted avg       0.78      0.78      0.78      5420
```



In [189]:

```
probs_adb_test = abcl.predict_proba(X_test_adb)
probs_adb_test = probs_adb_test[:,1]
auc_test_adb = roc_auc_score(y_test_adb, probs_adb_test)
print('The AUC score for AdaBoosting testing set is: %.3f'%auc_test_adb)

test_fpr_adb, test_tpr_adb, test_thresholds_adb = roc_curve(y_test_adb, probs_adb_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_adb, test_tpr_adb);
```

```
The AUC score for AdaBoosting testing set is: 0.860
```



Gradient Boosting

In [190]:

```
X_train_gdb, X_test_gdb, y_train_gdb, y_test_gdb = train_test_split(X_sm, y_sm, test_siz
e=0.3, random_state=42)

gbcl = GradientBoostingClassifier(n_estimators=50,random_state=1)
gbcl = gbcl.fit(X_train_gdb, y_train_gdb)
```

In [191]:

```
print('The model score for GradientBoosting training set is',gbcl.score(X_train_gdb,y_tra
in_gdb))
print('\n')
print('The model score for GradientBoosting testing set is',gbcl.score(X_test_gdb,y_test_
gdb))
```

```
The model score for GradientBoosting training set is 0.8260044289781715


The model score for GradientBoosting testing set is 0.8160516605166052
```

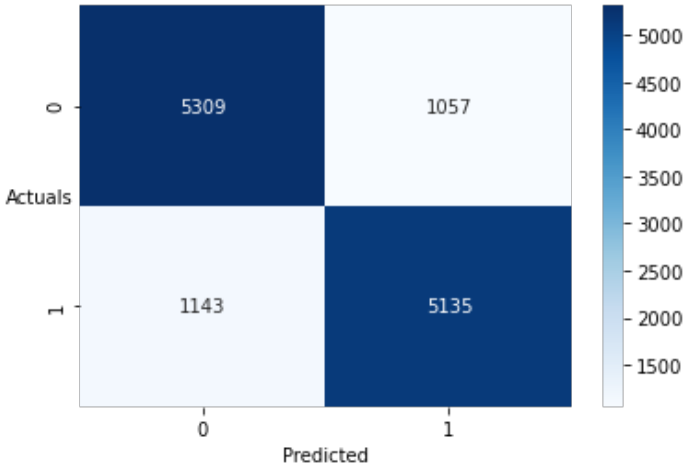
In [192]:

```
y_train_pred_gdb = gbcl.predict(X_train_gdb)
y_test_pred_gdb  = gbcl.predict(X_test_gdb)
```

In [193]:

```
print('The classification report for Gradientboosting training set is\n',classification_r
eport(y_train_gdb, y_train_pred_gdb))

sns.heatmap((confusion_matrix(y_train_gdb,y_train_pred_gdb)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

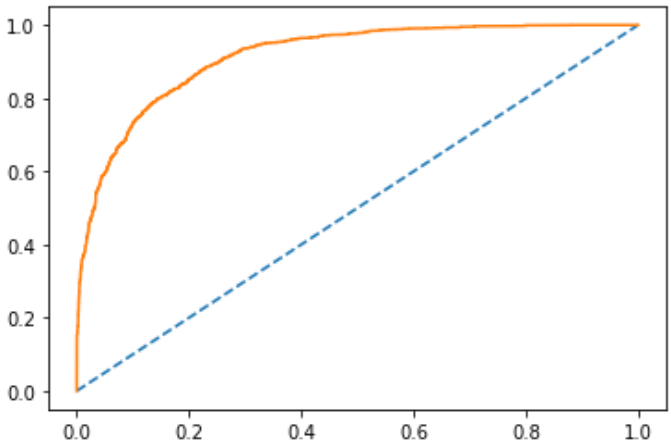
```
The classification report for Gradientboosting training set is
              precision    recall  f1-score   support

         0.0       0.82      0.83      0.83      6366
         1.0       0.83      0.82      0.82      6278

    accuracy                           0.83     12644
   macro avg       0.83      0.83      0.83     12644
weighted avg       0.83      0.83      0.83     12644
```



In [194]:

```
probs_gdb_train = gbcl.predict_proba(X_train_gdb)
probs_gdb_train = probs_gdb_train[:,1]
auc_train_gdb = roc_auc_score(y_train_gdb, probs_gdb_train)
print('The AUC score for GradientBoosting training set is: %.3f'%auc_train_gdb)

train_fpr_gdb, train_tpr_gdb, train_thresholds_gdb = roc_curve(y_train_gdb, probs_gdb_tra
in);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(train_fpr_gdb, train_tpr_gdb);
```

```
The AUC score for GradientBoosting training set is: 0.915
```

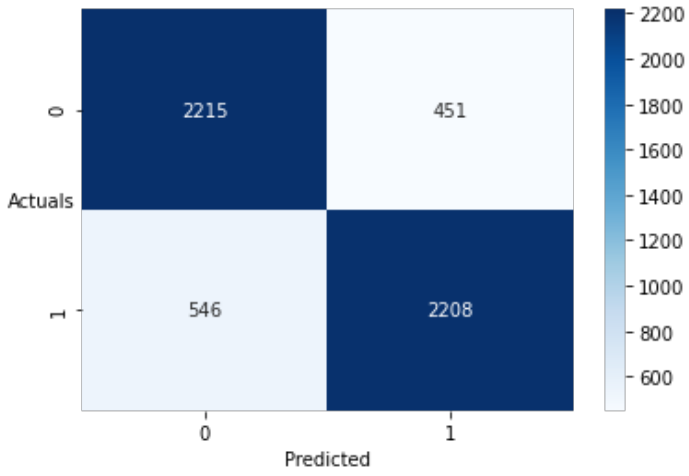


In [195]:

```
print('The classification report for Gradientboosting testing set is\n',classification_re
port(y_test_gdb, y_test_pred_gdb))

sns.heatmap((confusion_matrix(y_test_gdb,y_test_pred_gdb)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
```

```
The classification report for Gradientboosting testing set is
              precision    recall  f1-score   support

         0.0       0.80      0.83      0.82      2666
         1.0       0.83      0.80      0.82      2754
```

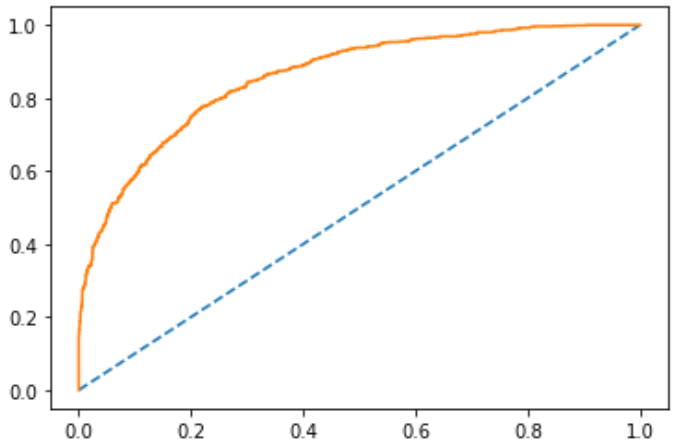
```
    accuracy                           0.82      5420
   macro avg       0.82      0.82      0.82      5420
weighted avg       0.82      0.82      0.82      5420
```



In [196]:

```
probs_gdb_test = abcl.predict_proba(X_test_gdb)
probs_gdb_test = probs_gdb_test[:,1]
auc_test_gdb = roc_auc_score(y_test_gdb, probs_gdb_test)
print('The AUC score for GradientBoosting testing set is: %.3f'%auc_test_gdb)

test_fpr_gdb, test_tpr_gdb, test_thresholds_gdb = roc_curve(y_test_gdb, probs_gdb_test);
plt.plot([0,1],[0,1], linestyle = '--');
plt.plot(test_fpr_gdb, test_tpr_gdb);
```

```
The AUC score for GradientBoosting testing set is: 0.860
```



In [197]:

```
LR_Train_class = classification_report(y_train_LR, y_train_pred_LR , output_dict=True)
df_LR_Train    = pd.DataFrame(LR_Train_class).transpose()

LR_Train_precision = round(df_LR_Train.loc['1.0'][0] , 3)
LR_Train_recall    = round(df_LR_Train.loc['1.0'][1] , 3)
LR_Train_F1score   = round(df_LR_Train.loc['1.0'][2] , 3)
LR_Train_Accuracy  = round(df_LR_Train.loc['accuracy'][0], 3)
LR_Train_auc_score = round(auc_train , 3)
```

In [198]:

```
LR_Test_class = classification_report(y_test_LR, y_test_pred_LR , output_dict=True)
df_LR_Test    = pd.DataFrame(LR_Test_class).transpose()

LR_Test_precision = round(df_LR_Test.loc['1.0'][0] , 3)
LR_Test_recall    = round(df_LR_Test.loc['1.0'][1] , 3)
LR_Test_F1score   = round(df_LR_Test.loc['1.0'][2] , 3)
LR_Test_Accuracy  = round(df_LR_Test.loc['accuracy'][0], 3)
LR_Test_auc_score = round(auc_test , 3)
```

In [199]:

```
LDA_Train_class = classification_report(y_train_LDA, y_train_pred_LDA , output_dict=True
)
df_LDA_Train    = pd.DataFrame(LDA_Train_class).transpose()

LDA_Train_precision = round(df_LDA_Train.loc['1.0'][0] , 3)
LDA_Train_recall    = round(df_LDA_Train.loc['1.0'][1] , 3)
LDA_Train_F1score   = round(df_LDA_Train.loc['1.0'][2] , 3)
LDA_Train_Accuracy  = round(df_LDA_Train.loc['accuracy'][0], 3)
LDA_Train_auc_score = round(auc_train_LDA , 3)
```

In [223]:

```
LDA_Test_class = classification_report(y_test_LDA, y_test_pred_LDA , output_dict=True)
df_LDA_Test    = pd.DataFrame(LDA_Test_class).transpose()

LDA_Test_precision = round(df_LDA_Test.loc['1.0'][0] , 3)
LDA_Test_recall    = round(df_LDA_Test.loc['1.0'][1] , 3)
LDA_Test_F1score   = round(df_LDA_Test.loc['1.0'][2] , 3)
LDA_Test_Accuracy  = round(df_LDA_Test.loc['accuracy'][0], 3)
LDA_Test_auc_score = round(auc_test_LDA , 3)
```

In [201]:

```
KNN_Train_class = classification_report(y_train_KNN, y_train_pred_KNN , output_dict=True
)
df_KNN_Train    = pd.DataFrame(KNN_Train_class).transpose()

KNN_Train_precision = round(df_KNN_Train.loc['1.0'][0] , 3)
KNN_Train_recall    = round(df_KNN_Train.loc['1.0'][1] , 3)
KNN_Train_F1score   = round(df_KNN_Train.loc['1.0'][2] , 3)
KNN_Train_Accuracy  = round(df_KNN_Train.loc['accuracy'][0], 3)
KNN_Train_auc_score = round(auc_train_KNN , 3)
```

In [202]:

```
KNN_Test_class = classification_report(y_test_KNN, y_test_pred_KNN , output_dict=True)
df_KNN_Test    = pd.DataFrame(KNN_Test_class).transpose()

KNN_Test_precision = round(df_KNN_Test.loc['1.0'][0] , 3)
KNN_Test_recall    = round(df_KNN_Test.loc['1.0'][1] , 3)
KNN_Test_F1score   = round(df_KNN_Test.loc['1.0'][2] , 3)
KNN_Test_Accuracy  = round(df_KNN_Test.loc['accuracy'][0], 3)
KNN_Test_auc_score = round(auc_test_KNN , 3)
```

In [203]:

```
NB_Train_class = classification_report(y_train_NB, y_train_pred_NB , output_dict=True)
df_NB_Train    = pd.DataFrame(NB_Train_class).transpose()

NB_Train_precision = round(df_NB_Train.loc['1.0'][0] , 3)
NB_Train_recall    = round(df_NB_Train.loc['1.0'][1] , 3)
NB_Train_F1score   = round(df_NB_Train.loc['1.0'][2] , 3)
NB_Train_Accuracy  = round(df_NB_Train.loc['accuracy'][0], 3)
NB_Train_auc_score = round(auc_train_NB , 3)
```

In [204]:

```
NB_Test_class = classification_report(y_test_NB, y_test_pred_NB , output_dict=True)
df_NB_Test    = pd.DataFrame(NB_Test_class).transpose()

NB_Test_precision = round(df_NB_Test.loc['1.0'][0] , 3)
NB_Test_recall    = round(df_NB_Test.loc['1.0'][1] , 3)
NB_Test_F1score   = round(df_NB_Test.loc['1.0'][2] , 3)
NB_Test_Accuracy  = round(df_NB_Test.loc['accuracy'][0], 3)
NB_Test_auc_score = round(auc_test_NB , 3)
```

In [205]:

```
DT_Train_class = classification_report(y_train_DT, y_train_pred_DT , output_dict=True)
df_DT_Train    = pd.DataFrame(DT_Train_class).transpose()

DT_Train_precision = round(df_DT_Train.loc['1.0'][0] , 3)
DT_Train_recall    = round(df_DT_Train.loc['1.0'][1] , 3)
DT_Train_F1score   = round(df_DT_Train.loc['1.0'][2] , 3)
DT_Train_Accuracy  = round(df_DT_Train.loc['accuracy'][0], 3)
DT_Train_auc_score = round(auc_train_DT, 3)
```

In [206]:

```
DT_Test_class = classification_report(y_test_DT, y_test_pred_DT , output_dict=True)
df_DT_Test    = pd.DataFrame(DT_Test_class).transpose()

DT_Test_precision = round(df_DT_Test.loc['1.0'][0] , 3)
DT_Test_recall    = round(df_DT_Test.loc['1.0'][1] , 3)
DT_Test_F1score   = round(df_DT_Test.loc['1.0'][2] , 3)
DT_Test_Accuracy  = round(df_DT_Test.loc['accuracy'][0], 3)
DT_Test_auc_score = round(auc_test_DT, 3)
```

In [207]:

```
RFC_Train_class = classification_report(y_train_RFC, y_train_pred_RFC , output_dict=True
)
df_RFC_Train    = pd.DataFrame(RFC_Train_class).transpose()

RFC_Train_precision = round(df_RFC_Train.loc['1.0'][0] , 3)
RFC_Train_recall    = round(df_RFC_Train.loc['1.0'][1] , 3)
RFC_Train_F1score   = round(df_RFC_Train.loc['1.0'][2] , 3)
RFC_Train_Accuracy  = round(df_RFC_Train.loc['accuracy'][0], 3)
RFC_Train_auc_score = round(auc_train_RFC, 3)
```

In [208]:

```
RFC_Test_class = classification_report(y_test_RFC, y_test_pred_RFC , output_dict=True)
df_RFC_Test    = pd.DataFrame(RFC_Test_class).transpose()

RFC_Test_precision = round(df_RFC_Test.loc['1.0'][0] , 3)
RFC_Test_recall    = round(df_RFC_Test.loc['1.0'][1] , 3)
RFC_Test_F1score   = round(df_RFC_Test.loc['1.0'][2] , 3)
RFC_Test_Accuracy  = round(df_RFC_Test.loc['accuracy'][0], 3)
RFC_Test_auc_score = round(auc_test_RFC, 3)
```

In [209]:

```
bg_Train_class = classification_report(y_train_bg, y_train_pred_bg , output_dict=True)
df_bg_Train    = pd.DataFrame(bg_Train_class).transpose()

bg_Train_precision = round(df_bg_Train.loc['1.0'][0] , 3)
bg_Train_recall    = round(df_bg_Train.loc['1.0'][1] , 3)
bg_Train_F1score   = round(df_bg_Train.loc['1.0'][2] , 3)
bg_Train_Accuracy  = round(df_bg_Train.loc['accuracy'][0], 3)
bg_Train_auc_score = round(auc_train_bg, 3)
```

In [210]:

```
bg_Test_class = classification_report(y_test_bg, y_test_pred_bg , output_dict=True)
df_bg_Test    = pd.DataFrame(bg_Test_class).transpose()

bg_Test_precision = round(df_bg_Test.loc['1.0'][0] , 3)
bg_Test_recall    = round(df_bg_Test.loc['1.0'][1] , 3)
bg_Test_F1score   = round(df_bg_Test.loc['1.0'][2] , 3)
bg_Test_Accuracy  = round(df_bg_Test.loc['accuracy'][0], 3)
bg_Test_auc_score = round(auc_test_bg, 3)
```

In [211]:

```
adb_Train_class = classification_report(y_train_adb, y_train_pred_adb , output_dict=True
)
df_adb_Train    = pd.DataFrame(adb_Train_class).transpose()
```

```
adb_Train_precision = round(df_adb_Train.loc['1.0'][0] , 3)
adb_Train_recall    = round(df_adb_Train.loc['1.0'][1] , 3)
adb_Train_F1score   = round(df_adb_Train.loc['1.0'][2] , 3)
adb_Train_Accuracy  = round(df_adb_Train.loc['accuracy'][0], 3)
adb_Train_auc_score = round(auc_train_adb, 3)
```

In [212]:

```
adb_Test_class = classification_report(y_test_adb, y_test_pred_adb , output_dict=True)
df_adb_Test    = pd.DataFrame(adb_Test_class).transpose()

adb_Test_precision = round(df_adb_Test.loc['1.0'][0] , 3)
adb_Test_recall    = round(df_adb_Test.loc['1.0'][1] , 3)
adb_Test_F1score   = round(df_adb_Test.loc['1.0'][2] , 3)
adb_Test_Accuracy  = round(df_adb_Test.loc['accuracy'][0], 3)
adb_Test_auc_score = round(auc_test_adb, 3)
```

In [213]:

```
gdb_Train_class = classification_report(y_train_gdb, y_train_pred_gdb , output_dict=True
)
df_gdb_Train    = pd.DataFrame(gdb_Train_class).transpose()

gdb_Train_precision = round(df_gdb_Train.loc['1.0'][0] , 3)
gdb_Train_recall    = round(df_gdb_Train.loc['1.0'][1] , 3)
gdb_Train_F1score   = round(df_gdb_Train.loc['1.0'][2] , 3)
gdb_Train_Accuracy  = round(df_gdb_Train.loc['accuracy'][0], 3)
gdb_Train_auc_score = round(auc_train_gdb, 3)
```

In [214]:

```
gdb_Test_class = classification_report(y_test_gdb, y_test_pred_gdb , output_dict=True)
df_gdb_Test    = pd.DataFrame(gdb_Test_class).transpose()

gdb_Test_precision = round(df_gdb_Test.loc['1.0'][0] , 3)
gdb_Test_recall    = round(df_gdb_Test.loc['1.0'][1] , 3)
gdb_Test_F1score   = round(df_gdb_Test.loc['1.0'][2] , 3)
gdb_Test_Accuracy  = round(df_gdb_Test.loc['accuracy'][0], 3)
gdb_Test_auc_score = round(auc_test_gdb, 3)
```

In [224]:

```
pd.set_option('display.max_columns', None)
index = ['Precision','Recall','F1 Score','Accuracy','AUC Score']
pd.DataFrame({'LR Train' :[LR_Train_precision , LR_Train_recall ,LR_Train_F1score, LR_Tr
ain_Accuracy, LR_Train_auc_score],
              'LR Test':[LR_Test_precision, LR_Test_recall, LR_Test_F1score, LR_Test_Acc
uracy,LR_Test_auc_score],
              'LDA Train'  :[LDA_Train_precision, LDA_Train_recall, LDA_Train_F1score, L
DA_Train_Accuracy, LDA_Train_auc_score],
              'LDA Test' :[LDA_Test_precision,LDA_Test_recall,LDA_Test_F1score,LDA_Test_
Accuracy,LDA_Test_auc_score],
              'KNN Train' :[KNN_Train_precision , KNN_Train_recall ,KNN_Train_F1score, K
NN_Train_Accuracy, KNN_Train_auc_score],
              'KNN Test':[KNN_Test_precision, KNN_Test_recall, KNN_Test_F1score, KNN_Tes
t_Accuracy,KNN_Test_auc_score],
              'NB Train'  :[NB_Train_precision, NB_Train_recall, NB_Train_F1score, NB_Tr
ain_Accuracy, NB_Train_auc_score],
              'NB Test' :[NB_Test_precision, NB_Test_recall, NB_Test_F1score, NB_Test_Ac
curacy, NB_Test_auc_score],
              'CART Train':[DT_Train_precision , DT_Train_recall ,DT_Train_F1score, DT_T
rain_Accuracy, DT_Train_auc_score],
              'CART Test':[DT_Test_precision, DT_Test_recall, DT_Test_F1score, DT_Test_A
ccuracy,DT_Test_auc_score],
              'RFC Train':[RFC_Train_precision , RFC_Train_recall ,RFC_Train_F1score, RF
C_Train_Accuracy, RFC_Train_auc_score],
              'RFC Test':[RFC_Test_precision, RFC_Test_recall, RFC_Test_F1score, RFC_Tes
t_Accuracy,RFC_Test_auc_score],
              'Bagging Train':[bg_Train_precision , bg_Train_recall ,bg_Train_F1score, b
g_Train_Accuracy, bg_Train_auc_score],
```

```
                'Bagging Test':[bg_Test_precision, bg_Test_recall, bg_Test_F1score, bg_Tes
t_Accuracy,bg_Test_auc_score],
                'Ada Boosting Train':[adb_Train_precision, adb_Train_recall, adb_Train_F1s
core, adb_Train_Accuracy, adb_Train_auc_score],
                'Ada Boosting Test':[adb_Test_precision,adb_Test_recall,adb_Test_F1score,a
db_Test_Accuracy,adb_Test_auc_score],
                'Gradient Boosting Train':[gdb_Train_precision, gdb_Train_recall, gdb_Trai
n_F1score, gdb_Train_Accuracy, gdb_Train_auc_score],
                'Gradient Boosting Test':[gdb_Test_precision,gdb_Test_recall,gdb_Test_F1sc
ore,gdb_Test_Accuracy,gdb_Test_auc_score],},index= index)
```

Out[224]:

	LR Train	LR Test	LDA Train	LDA Test	KNN Train	KNN Test	NB Train	NB Test	CART Train	CART Test	RFC Train	RFC Test	Bagging Train	Bagging Test	Ada Boosting Train	Boos
Precision	0.714	0.730	0.713	0.730	0.987	0.976	0.664	0.668	1.0	0.983	1.0	0.997	1.0	0.995	0.781	0
Recall	0.737	0.736	0.737	0.734	0.999	0.997	0.763	0.761	1.0	0.980	1.0	0.992	1.0	0.989	0.773	0
F1 Score	0.725	0.733	0.725	0.732	0.993	0.986	0.710	0.711	1.0	0.981	1.0	0.995	1.0	0.992	0.777	0
Accuracy	0.723	0.727	0.722	0.727	0.993	0.986	0.691	0.686	1.0	0.981	1.0	0.995	1.0	0.992	0.780	0
AUC Score	0.771	0.777	0.770	0.776	1.000	0.999	0.749	0.753	1.0	0.981	1.0	1.000	1.0	0.999	0.865	0