```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import mutual_info_classif as mic
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler
```

```
df = pd.read_csv(r'customer_booking.csv', encoding="ISO-8859-1")
```

```
df.head()
```

ur	flight_day	route	booking_origin	wants_
7	Sat	AKLDEL	New Zealand	
3	Sat	AKLDEL	New Zealand	
17	Wed	AKLDEL	India	
4	Sat	AKLDEL	New Zealand	
15	Wed	AKLDEL	India	

Next steps: Generate code with df View recommended plots

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   num_passengers         50000 non-null  int64
 1   sales_channel          50000 non-null  object
 2   trip_type              50000 non-null  object
 3   purchase_lead          50000 non-null  int64
 4   length_of_stay         50000 non-null  int64
 5   flight_hour            50000 non-null  int64
 6   flight_day             50000 non-null  object
 7   route                  50000 non-null  object
 8   booking_origin         50000 non-null  object
 9   wants_extra_baggage    50000 non-null  int64
 10  wants_preferred_seat   50000 non-null  int64
 11  wants_in_flight_meals  50000 non-null  int64
 12  flight_duration        50000 non-null  float64
 13  booking_complete       50000 non-null  int64
dtypes: float64(1), int64(8), object(5)
memory usage: 5.3+ MB
```

```
df.isnull().sum()
```

```
num_passengers           0
sales_channel            0
trip_type                0
purchase_lead            0
length_of_stay           0
flight_hour              0
flight_day               0
route                    0
booking_origin           0
wants_extra_baggage      0
wants_preferred_seat     0
wants_in_flight_meals    0
flight_duration          0
booking_complete         0
dtype: int64
```

```
df['flight_day'].unique()
```

```
array(['Sat', 'Wed', 'Thu', 'Mon', 'Sun', 'Tue', 'Fri'], dtype=object)
```

```
mapping = {
    "Mon": 1,
    "Tue": 2,
    "Wed": 3,
    "Thu": 4,
    "Fri": 5,
    "Sat": 6,
    "Sun": 7
}

df['flight_day'] = df['flight_day'].map(mapping)
```

```
df['flight_day'].unique()
```

```
array([6, 3, 4, 1, 7, 2, 5])
```

```
df.describe()
```

	num_passengers	purchase_lead	length_of_stay	flight_hour	flight_day	wants_extra_baggage	wants_preferred_seat	wants_in
count	50000.000000	50000.000000	50000.00000	50000.00000	50000.000000	50000.000000	50000.000000	
mean	1.591240	84.940480	23.04456	9.06634	3.814420	0.668780	0.296960	
std	1.020165	90.451378	33.88767	5.41266	1.992792	0.470657	0.456923	
min	1.000000	0.000000	0.00000	0.00000	1.000000	0.000000	0.000000	
25%	1.000000	21.000000	5.00000	5.00000	2.000000	0.000000	0.000000	
50%	1.000000	51.000000	17.00000	9.00000	4.000000	1.000000	0.000000	
75%	2.000000	115.000000	28.00000	13.00000	5.000000	1.000000	1.000000	
max	9.000000	867.000000	778.00000	23.00000	7.000000	1.000000	1.000000	

```
# Categorical Columns
cat_cols=df.select_dtypes("object")
```

```
# Checking the unique values in categorical columns
for col in cat_cols:
  print("\nUnique values for column '{}':".format(col))
  print(df[col].unique(), "\nUnique count: {}".format(df[col].nunique()))
```

```
 [ New Zealand   India   United Kingdom   China   South Korea   Japan
  'Malaysia' 'Singapore' 'Switzerland' 'Germany' 'Indonesia'
  'Czech Republic' 'Vietnam' 'Thailand' 'Spain' 'Romania' 'Ireland' 'Italy'
  'Slovakia' 'United Arab Emirates' 'Tonga' 'Réunion' '(not set)'
  'Saudi Arabia' 'Netherlands' 'Qatar' 'Hong Kong' 'Philippines'
  'Sri Lanka' 'France' 'Croatia' 'United States' 'Laos' 'Hungary'
  'Portugal' 'Cyprus' 'Australia' 'Cambodia' 'Poland' 'Belgium' 'Oman'
  'Bangladesh' 'Kazakhstan' 'Brazil' 'Turkey' 'Kenya' 'Taiwan' 'Brunei'
  'Chile' 'Bulgaria' 'Ukraine' 'Denmark' 'Colombia' 'Iran' 'Bahrain'
  'Solomon Islands' 'Slovenia' 'Mauritius' 'Nepal' 'Russia' 'Kuwait'
  'Mexico' 'Sweden' 'Austria' 'Lebanon' 'Jordan' 'Greece' 'Mongolia'
  'Canada' 'Tanzania' 'Peru' 'Timor-Leste' 'Argentina' 'New Caledonia'
  'Macau' 'Myanmar (Burma)' 'Norway' 'Panama' 'Bhutan' 'Norfolk Island'
  'Finland' 'Nicaragua' 'Maldives' 'Egypt' 'Israel' 'Tunisia'
  'South Africa' 'Papua New Guinea' 'Paraguay' 'Estonia' 'Seychelles'
  'Afghanistan' 'Guam' 'Czechia' 'Malta' 'Vanuatu' 'Belarus' 'Pakistan'
  'Iraq' 'Ghana' 'Gibraltar' 'Guatemala' 'Algeria' 'Svalbard & Jan Mayen']
Unique count: 104
```

```
# Label Encoding the categorical variables.
label_encode = LabelEncoder()

for col in cat_cols:
  df[col]=label_encode.fit_transform(df[col])
  print("\nUnique values for column '{}':".format(col))
  print(df[col].unique(), "\nUnique count: {}".format(df[col].nunique()))
```

```
Unique count: 3

Unique values for column 'route':
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  17  18  19
  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  36  37  38
  39  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57
  58  59  60  61  62  64  65  66  67  68  69  70  71  72  73  74  75  76
  77  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95
  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113
 114 115 116 117 118 119 121 122 125 126 127 129 130 131 132 133 134 136
 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154
 155 157 158 159 160 161 162 163 165 166 167 170 171 172 173 174 175 176
 177 178 179 180 181 182 183 185 187 188 189 190 192 193 194 195 196 197
 198 199 200 202 203 204 205 207 208 209 210 212 213 214 217 218 220 221
 222 223 224 226 228 230 231 232 233 234 236 237 238 239 240 241 243 245
 247 248 249 250 251 252 253 254 255 256 257 258 260 261 262 263 264 265
 266 267 268 269 270 271 272 273 276 277 278 279 280 281 283 284 285 286
 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304
 305 306 307 308 309 310 311 313 314 315 316 317 318 319 320 321 322 323
 324 326 327 328 329 330 331 332 333 335 336 337 338 339 340 341 342 343
 344 345 347 348 349 350 351 354 355 357 359 361 362 363 364 365 366 367
 368 371 372 373 374 375 376 377 378 379 380 381 383 384 385 386 387 388
 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406
 407 408 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425
 427 428 429 430 431 432 433 434 435 436 437 438 440 442 443 444 445 446
 447 448 449 450 451 452 453 454 455 456 457 458 460 461 462 463 465 466
 467 468 469 470 471 472 474 475 476 477 478 479 480 481 482 483 484 485
 486 487 488 489 490 491 492 493 494 496 497 498 502 503 504 505 506 508
 509 513 514 515 516 517 518 519 520 521 522 523 525 526 527 528 529 530
 531 533 534 536 537 538 540 541 542 543 544 545 546 547 548 549 550 551
 552 553 555 556 558 559 560 561 562 563 564 565 566 568 569 570 571 572
 573 574 575 576 577 578 579 580 581 582 583 584 586 587 588 590 591 593
 594 595 596 597 600 601 603 604 605 606 607 608 609 610 611 612 613 614
 615 616 617 618 619 620 621 622 623 625 626 627 628 629 630 631 632 633
 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 652
 653 655 657 658 659 660 661 662 663 664 665 666 667 668 669 670 673 674
 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692
 693 694 695 696 697 698 699 700 701 702 703 704 705 707 708 709 710 711
 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 730
 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748
 749 750 751 752 754 755 756 757 758 759 760 761 762 763 764 765 767 768
 770 771 772 773 774 775 776 777 778 780 781 782 783 784 785 786 787 788
 789 790 793 794 795 797 352 439 473 500 510 554 589 753 766 274 312 360
 501 507 535 592 599 602 706 729 791  35  40 135 211 215 225 242 244 275
 334 356 382 441 495 524 557 567 656 769 779 796  16  63  78 123 124 156
 164 191 201 206 216 259 282 325 353 358 409 459 511 512 532 598 624 651
 671 792 798  15 120 128 168 169 184 186 219 227 229 235 246 346 369 370
 426 464 499 539 585 654 672]
Unique count: 799

Unique values for column 'booking_origin':
[ 61  36 100  17  85  43  51  80  90  28  37  21 103  93  86  75  40  42
  81  99  95  77   0  78  59  74  34  71  87  27  19 101  48  35  73  20
   4  14  72   9  65   7  45  11  97  46  91  12  16  13  98  23  18  38
   6  83  82  54  58  76  47  55  89   5  49  44  31  56  15  92  70  94
   3  60  50  57  64  67  10  63  26  62  52  24  41  96  84  68  69  25
  79   1  32  22  53 102   8  66  39  29  30  33   2  88]
Unique count: 104
```

```
# Checking the datatypes of converted columns.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   num_passengers         50000 non-null  int64
 1   sales_channel          50000 non-null  int64
 2   trip_type              50000 non-null  int64
 3   purchase_lead          50000 non-null  int64
 4   length_of_stay         50000 non-null  int64
 5   flight_hour            50000 non-null  int64
 6   flight_day             50000 non-null  int64
 7   route                  50000 non-null  int64
 8   booking_origin         50000 non-null  int64
 9   wants_extra_baggage    50000 non-null  int64
 10  wants_preferred_seat   50000 non-null  int64
 11  wants_in_flight_meals  50000 non-null  int64
 12  flight_duration        50000 non-null  float64
 13  booking_complete       50000 non-null  int64
dtypes: float64(1), int64(13)
memory usage: 5.3 MB
```

```
df.describe()
```

	num_passengers	sales_channel	trip_type	purchase_lead	length_of_stay	flight_hour	flight_day	route	booking_c
count	50000.000000	50000.000000	50000.000000	50000.000000	50000.00000	50000.00000	50000.000000	50000.000000	50000.0
mean	1.591240	0.112360	1.987620	84.940480	23.04456	9.06634	3.814420	391.905800	38.2
std	1.020165	0.315812	0.129873	90.451378	33.88767	5.41266	1.992792	227.297259	32.7
min	1.000000	0.000000	0.000000	0.000000	0.00000	0.00000	1.000000	0.000000	0.0
25%	1.000000	0.000000	2.000000	21.000000	5.00000	5.00000	2.000000	203.000000	4.0
50%	1.000000	0.000000	2.000000	51.000000	17.00000	9.00000	4.000000	381.000000	37.0
75%	2.000000	0.000000	2.000000	115.000000	28.00000	13.00000	5.000000	611.000000	57.0
max	9.000000	1.000000	2.000000	867.000000	778.00000	23.00000	7.000000	798.000000	103.0

```
# Converting the dataset into features and label.
X = df.drop('booking_complete', axis=1)
y = df['booking_complete']
```

```
# Calculating the Mutual Information Scores.
fi_scores = mic(X, y)
fi_df = pd.DataFrame({"Columns": X.columns, "Feature_Importance_Score": fi_scores})
fi_df = fi_df.sort_values(by="Feature_Importance_Score", ascending=False)
fi_df
```

1 to 13 of 13 entries Filter

index	Columns	Feature_Importance_Score
7	route	0.054329262414762436
8	booking_origin	0.04066682193164639
12	flight_duration	0.0178521002368639
2	trip_type	0.00861499611598715
9	wants_extra_baggage	0.008540985294009928
4	length_of_stay	0.004678082186576393
10	wants_preferred_seat	0.003252846641099260 7
11	wants_in_flight_meals	0.0024704867489013793
5	flight_hour	0.0021878994957127418
3	purchase_lead	0.0018886673251024
0	num_passengers	3.793555530418047e-05
1	sales_channel	0.0
6	flight_day	0.0

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

Next steps: Generate code with fi_df View recommended plots

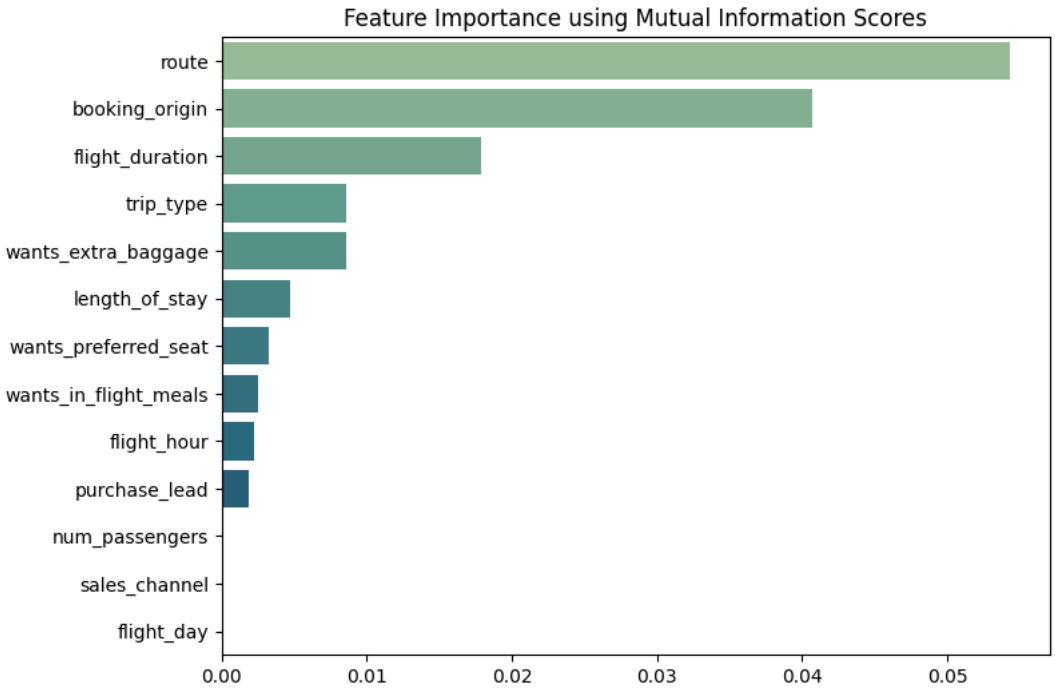
```
# DATA VISUALIZATION- FEATURE IMPORTANCE BARPLOT

plt.figure(figsize=(8, 6))
sns.barplot(x="Feature_Importance_Score", y="Columns", data=fi_df, palette="crest")
plt.title("Feature Importance using Mutual Information Scores")
plt.xlabel("")
plt.ylabel("")
plt.show()
```

```
<ipython-input-24-f4394e07f5e0>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `le

  sns.barplot(x="Feature_Importance_Score", y="Columns", data=fi_df, palette="crest")
```




```
# MODEL TRAINING AND CROSS VALIDATION

# Function to split the data into training and validation set.
def train_val_split(X, y):
    train_X, val_X, train_y, val_y = train_test_split(X, y, test_size=0.2, random_state=7)
    return train_X, val_X, train_y, val_y

# Function to select the top-n or all the features from data.
def selecting_top_n_or_all_features(n=5):
    if str(n).lower() == "all":
        X = df[list(fi_df.Columns)]
    else:
        X = df[list(fi_df.Columns[:n])]

# One-Hot-Encoding the variables which were Categorical variables prior to Label-Encoding.
    for col in X.select_dtypes("int32"):
        X = pd.get_dummies(X, columns=[col])
    return X

# Function to fit the data on RandomForestClassifier and product training and validation scores.
def fit_rfc(top_n):
    X = selecting_top_n_or_all_features(top_n)
    train_X, val_X, train_y, val_y = train_val_split(X, y)

 # Normalizing the Dataset.
    scaler = MinMaxScaler()
    train_X = scaler.fit_transform(train_X)
    val_X = scaler.transform(val_X)

    model = RandomForestClassifier(random_state=7)
    model.fit(train_X, train_y)

    train_y_pred = model.predict(train_X)
    val_y_pred = model.predict(val_X)

    print("Training Accuracy Score:", accuracy_score(train_y, train_y_pred))
    print("Validation Accuracy Score:", accuracy_score(val_y, val_y_pred))
```

```
print("\n===== Model Evaluation for Top-6 features =====\n")
fit_rfc(6)
```

```
===== Model Evaluation for Top-6 features =====

Training Accuracy Score: 0.907575
Validation Accuracy Score: 0.8301
```

```
print("\n===== Model Evaluation for All features =====\n")
fit_rfc("all")
```

```
===== Model Evaluation for All features =====

Training Accuracy Score: 0.999825
Validation Accuracy Score: 0.8533
```

Start coding or generate with AI.