פרויקט כריית מידע וייצוג מידע

פרויקט סוף

תאריך: 21\7\2024

מגישים: יקיר מרמלשטיין ושהם קראוס
ת.ז: 325694081, 207231937

<u>ראשי פרקים</u>

Case problem and mission
Data information
Data cleaning
Data statistics
Data correlation and visualization
Data manipulation
Data preprocessing
Data modeling and hyperparameters
Data training and model evaluation
Test Data preprocessing and prediction

Case problem and mission

תיאור הבעיה:

בתי מלון לעתים קרובות מתמודדים עם אתגרים בניהול הזמנות בשל ביטולים בלתי צפויים. ביטולים אלה עלולים להוביל לאובדן הכנסות והקצאת משאבים לא יעילה. חיזוי מדויק של ביטולי הזמנות הוא חיוני לבתי מלון כדי למטב את שיעורי התפוסה ולהגדיל את הרווחיות.

משימתנו:

בפרויקט זה, עלינו לנתח מערך נתונים של הזמנות מלון כדי לפתח מודל שיכול לחזות אילו הזמנות סביר שיבוטלו. זה יאפשר לבתי מלון ליישם אסטרטגיות הזמנה יעילות יותר ולשפר את היעילות התפעולית הכוללת.

תחילה נחקור וננתח את מערך הנתונים לאימון -'hotels_train' באמצעות שיטות סטטיסטיות וכלי ויזואליזציה, ונבצע עיבוד מקדים.

לאחר מכן, נאמן מודל סיווג על מערך הנתונים המעובד ונבצע תחזית עבור מערך הנתונים לבדיקה המצורף -'hotels_test'.

Dafa information

עבדנו בסביבת העבודה -Colab

תחילה ייבאנו את הספריות המתאימות הנוגעות לעיבוד נתונים,הצגת נתונים והפעלת המודלים השונים

ביניהן numpy ,pandas,matplotlib,sklearn,torch

לאחר מכן ייבאנו את הקבצי הדאטה למסמך העבודה

```
from google.colab import drive
#drive.mount('/content/drive')
train_df = pd.read_csv("/content/hotels_train.csv")
test_df = pd.read_csv("/content/hotels_test.csv")
```

כעת על מנת לעבוד עם הדאטה יש צורך להבין כיצד הדאטה מאורגן, לשם כך הצגנו את 5 הרשומות הראשונות בקובץ האימון באמצעות פקודת head.

בנוסף הדפסנו את מספר השורות -הרשומות בדאטה האימון ואת מספר העמודות- הפרמטרים ממנו הוא מורכב

קיבלנו מאגר נתונים המכיל כ18 עמודות וכ- 27213 שורות.

```
# presenting the number of rows and column in the data set
print("Shape: ", train_df.shape)
# presenting the 5 first rows of the data set
#in order to visually see the data we are working on
train_df.head()
```

Shape: (27213, 18)

	ID	weekend_nights	week_nights	room_type	board_type	n_adults	n_less_12	n_more_12	booked_tour	n_requests	lead_time	purchase_type	n_p_cacellation	n_p_not_cacellation	repeated	price	date	is_canceled
0	INN09588	1	5	Room_Type 1	half board	2	0	0	0	2	34.0	Online	0	0	0	108.4	11/28/2018	0
1	INN07691	0	3	Room_Type 1	NaN	2	0	0	0	0	365.0	NaN	0	0	0	NaN	11/03/2018	1
2	INN32192	0	2	Room_Type 4	half board	1	0	0	0	1	148.0	Online	0	0	0	137.3	05/06/2018	0
3	INN32218	1	2	Room_Type 1	NaN	2	0	0	0	0	502.0	Offline	0	0	0	127.0	9/26/2018	1
4	INN02994	1	3	Room_Type 4	half board	2	0	1	0	2	32.0	Offline	0	0	0	110.0	10/19/2017	0

כמו כן השתמשנו בפקודת Info בכדי להציג בצורה מסודרת את עמודות הטבלה כדי שנוכל לדעת את ה-Data type עבור כל עמודה בצורה נוחה .(בנוסף ניתן לראות כי חלק מהעמודות לא שלמות , נבחן זאת בהמשך באמצעות פונקציית isnull).

ניתן לראות כי סוגי העמודות הינם מסוג : float,int ,object .

```
#present the data columns ,their non_null values and their type
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27213 entries, 0 to 27212
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   ID                    27213 non-null  object
 1   weekend_nights        27213 non-null  int64
 2   week_nights           27213 non-null  int64
 3   room_type             27213 non-null  object
 4   board_type            19045 non-null  object
 5   n_adults              27213 non-null  int64
 6   n_less_12             27213 non-null  int64
 7   n_more_12             27213 non-null  int64
 8   booked_tour           27213 non-null  int64
 9   n_requests            27213 non-null  int64
 10  lead_time             26794 non-null  float64
 11  purchase_type         22366 non-null  object
 12  n_p_cacellation       27213 non-null  int64
 13  n_p_not_cacellation   27213 non-null  int64
 14  repeated              27213 non-null  int64
 15  price                 23808 non-null  float64
 16  date                  27213 non-null  object
 17  is_canceled           27213 non-null  int64
dtypes: float64(2), int64(11), object(5)
memory usage: 3.7+ MB
```

Data cleaning

1.הורדת העמודות הלא נחוצות:

- באמצעות פקודת drop הורדנו את העמודה של ה -ID

```
[ ]  train_df = train_df.drop(['ID'], axis=1)
```

2. טיפול בחוסר התאמות:

- זיהינו את התאריך '2018-2-29' תאריך שאינו קיים בשנה הקלאנדרית 2018 כיוון שמדובר מספר נמוך של רשומות מתחת ל12 , בחרנו להסיר אותן מקובץ הtrain

```
[ ]  train_df = train_df[train_df['date']!= '2018-2-29']
```

```
train_df.isnull().sum()

ID                        0
weekend_nights            0
week_nights               0
room_type                 0
board_type             8168
n_adults                  0
n_less_12                 0
n_more_12                 0
booked_tour               0
n_requests                0
lead_time               419
purchase_type          4847
n_p_cacellation           0
n_p_not_cacellation       0
repeated                  0
price                  3405
date                      0
is_canceled               0
dtype: int64
```

נבחן את כמות הערכים החסרים בדאטה האימון ביחס לעמודות באמצעות isnull

3. טיפול בערכים חסרים:

- בשלב הצגת הנתונים זיהינו חוסרים משמעותיים ב4 קטגוריות (עמודות)
- 'Board_type' - שחסרים לנו כ8168 ערכים השלמנו את הערכים באמצעות
- נשלים את הערכים החסרים ב-board_type באמצעות הערך הנפוץ ביותר בקבוצת purchase_type המתאימה.
- 'lead_time'- שחסרים לנו כ419 ערכים
- נשלים את הערכים החסרים ב-lead_time עם הערך הממוצע הכללי של העמודה lead_time.
- 'purchase_type' - שחסרים לנו כ4847 ערכים
- נשלים את הערכים החסרים ב-purchase_type באמצעות הערך הנפוץ ביותר בקבוצת board_type המתאימה.
- כאשר חסר הערך גם בעמודת ה-purchase_type וגם בעמודת ה-board_type נשלים כל ערך חסר שנותר בעמודות board_type ו-purchase_type עם הערכים הנפוצים ביותר שנשמרו קודם לכן.
- 'price' - שחסרים לנו כ3405 ערכים
- נשלים את הערכים החסרים ב-price באמצעות הערך הממוצע בקבוצת n_adults המתאימה.

```
overall_board_mode = train_df['board_type'].mode()[0]
overall_purchase_mode = train_df['purchase_type'].mode()[0]

# Fill missing values in 'board_type' with the mode for each 'purchase_type'
train_df['board_type'] = train_df.groupby('purchase_type', group_keys=False)['board_type'].apply(lambda x: x.fillna(x.mode()[0] if not x.mode().empty else '0'))

# Fill missing values in 'purchase_type' with the mode for each 'board_type'
train_df['purchase_type'] = train_df.groupby('board_type', group_keys=False)['purchase_type'].apply(lambda x: x.fillna(x.mode()[0] if not x.mode().empty else '1'))

# Fill missing values in 'price' with the mean for each 'n_adults'
train_df['price'] = train_df.groupby('n_adults', group_keys=False)['price'].apply(lambda x: x.fillna(x.mean()))

# Fill missing values in 'lead_time' with the median for all data
train_df['lead_time'] = train_df['lead_time'].fillna(train_df['lead_time'].median())

# Fill the data for line without board_type and purchase_type
train_df['board_type'] = train_df['board_type'].fillna(overall_board_mode)
train_df['purchase_type'] = train_df['purchase_type'].fillna(overall_purchase_mode)
```

Data statistics

```
# basic statistics of each column in the dataset
train_df.describe(include='all')
```

	ID	weekend_nights	week_nights	room_type	board_type	n_adults	n_less_12	n_more_12	booked_tour	n_requests	lead_time
count	27213	27213.000000	27213.000000	27213	19045	27213.000000	27213.000000	27213.000000	27213.000000	27213.000000	26794.000000
unique	27213	NaN	NaN	7	4	NaN	NaN	NaN	NaN	NaN	NaN
top	INN09588	NaN	NaN	Room_Type 1	half board	NaN	NaN	NaN	NaN	NaN	NaN
freq	1	NaN	NaN	21084	14591	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	0.812810	2.197332	NaN	NaN	1.845221	0.052989	0.053357	0.031750	0.621100	102.952377
std	NaN	0.869317	1.403576	NaN	NaN	0.519715	0.266150	0.268688	0.175336	0.785642	103.498942
min	NaN	0.000000	0.000000	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	NaN	0.000000	1.000000	NaN	NaN	2.000000	0.000000	0.000000	0.000000	0.000000	21.000000
50%	NaN	1.000000	2.000000	NaN	NaN	2.000000	0.000000	0.000000	0.000000	0.000000	69.000000
75%	NaN	2.000000	3.000000	NaN	NaN	2.000000	0.000000	0.000000	0.000000	1.000000	153.000000
max	NaN	7.000000	17.000000	NaN	NaN	4.000000	6.000000	4.000000	1.000000	5.000000	532.000000

הצגת טבלת סטטיסטיקה:

showing the columns names in a list:

```
numeric_col = train_df.describe().columns # to get the numeric column
numeric_col
```

```
Index(['weekend_nights', 'week_nights', 'n_adults', 'n_less_12', 'n_more_12',
       'booked_tour', 'n_requests', 'lead_time', 'n_p_cacellation',
       'n_p_not_cacellation', 'repeated', 'price', 'is_canceled'],
      dtype='object')
```

separating the dataset into numeric and numinal data:

```
numeric_data = data[numeric_col]
nominal_data = data.drop(numeric_col, axis=1)
```

הפרדנו את עמודות הדאטה
ל2 קטגוריות מספרי וקטגורי :

<u>צידדויות</u>

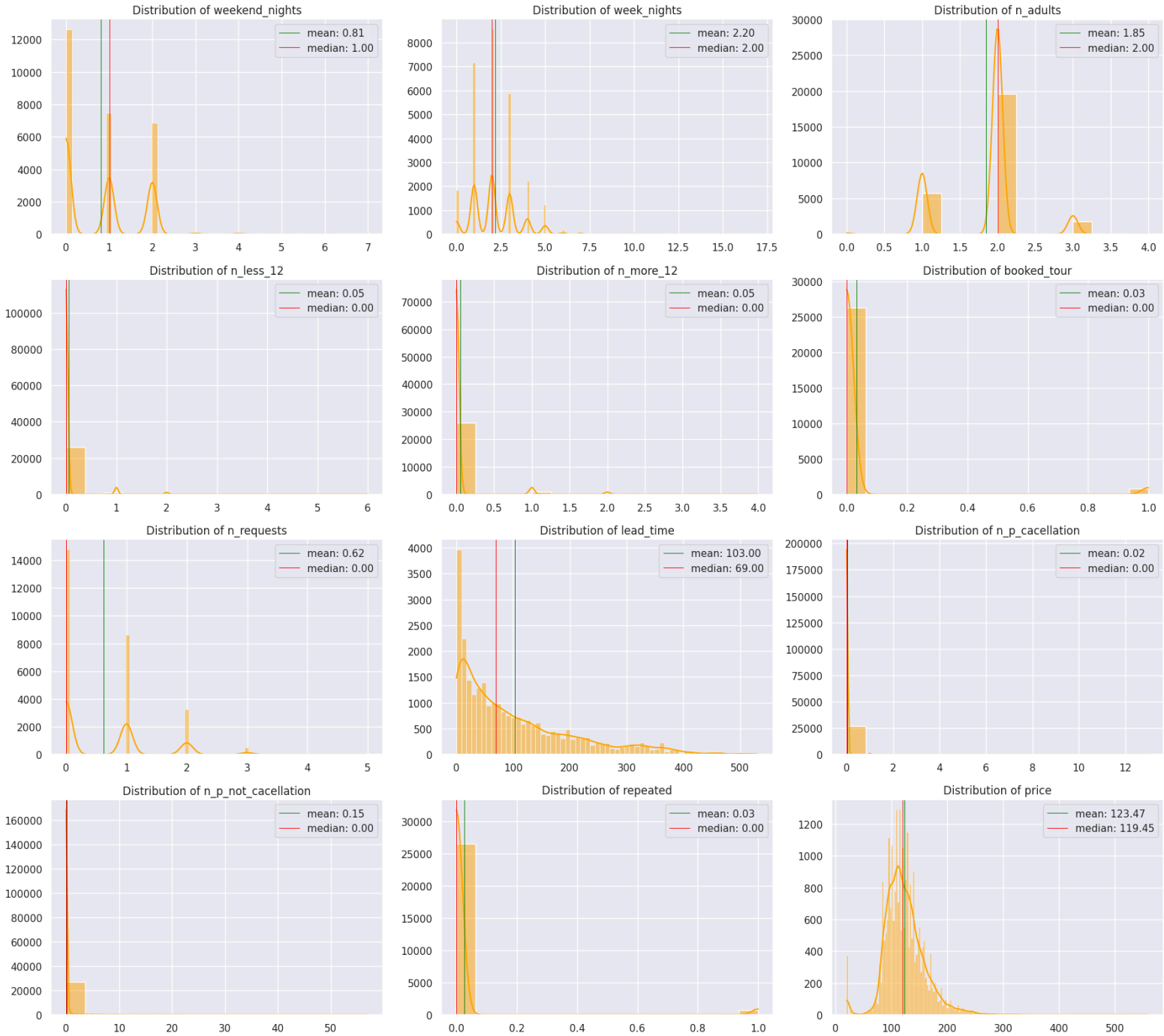
עבור הקטגוריות המספריות חישבנו את ה-Skewness

```
# Calculate skewness of numerical columns
numerical_columns = ['lead_time', 'price', 'week_nights', 'n_adults', 'weekend_nights', 'n_requests']
skewness = train_df[numerical_columns].skew()
print("Skewness before transformation:\n", skewness)
```

```
Skewness before transformation:
 lead_time         1.290752
price             0.682450
week_nights       1.543132
n_adults         -0.324842
weekend_nights    0.719113
n_requests        1.143249
dtype: float64
```

לצורך שלב הצגת הנתונים הסטטיסטים ביצענו ויזואליזציה שמשלבת בתוכה בצורה ויזואלית נראית לעין את גרפי ההתפלגות השונים עבור כל אחת העמודות בדאטה, בשילוב הצגת הממוצע והחציון עבור הדאטה באמצעות קווים אנכיים. כמו כן ניתן לראות דרך ההצגה את ההבדל בין הדאטה המספרי לדאטה הקטגורי .

כאשר בציר ה-Y מציין מספר המופעים של אותו ערך ברשומות הדאטה, ובציר -X הערך בעמודה עבור אותו מופע.
התפלגויות משולבת ממוצע וחציון



(בנוסף ביצענו אחד מובחן יותר ללא סימוני חציון וממוצע.)

Data correlation vizualization

עבור כלל העמודות בדאטה ביצענו הצגה ראשונית של קורלציה באמצעות ספריית Seaborn בHeat map correlation matrix

מטריצת קורלציה של Seaborn heatmap מספקת תובנות חשובות לגבי הקשרים בין מאפיינים במערך נתונים:

<u>זיהוי קשרים חזקים:</u> עוזרת בזיהוי מאפיינים בעלי קורלציה גבוהה שיכולים להעיד על כפילות מידע.

<u>זיהוי מולטיקולינאריות:</u> מזהה מאפיינים שמראים קורלציה גבוהה ביניהם, מה שיכול להוות בעיה עבור מודלים מסוימים.

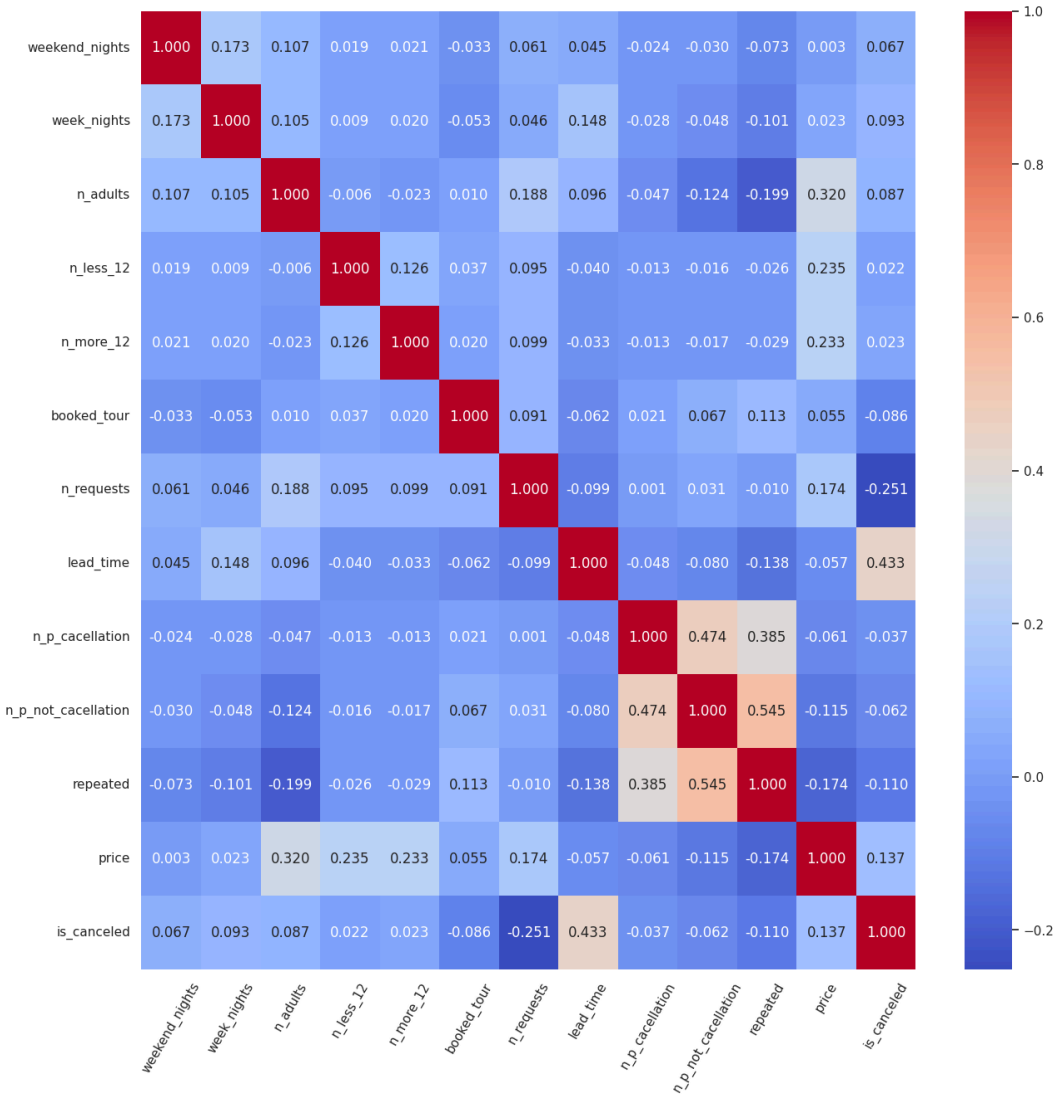
<u>בחירת מאפיינים:</u> מסייעת להחליט אילו מאפיינים לשמור או להסיר בהתבסס על הקורלציה שלהם עם המשתנה המוסבר.

<u>הנדסת מאפיינים:</u> מנחה ביצירת מאפיינים חדשים או בהסרת מאפיינים מיותרים.

<u>הבנת מבנה הנתונים:</u> מספקת תחושה ויזואלית של המבנה הכללי והתפלגות הקורלציות.

<u>ניקוי נתונים:</u>עוזרת לזהות שגיאות נתונים או חריגות ומדגישה את הצורך בהשלמת ערכים חסרים.

<u>תובנות מודל:</u> מספקת תובנות לגבי חשיבות המאפיינים והקשרים שהמודל מנצל.



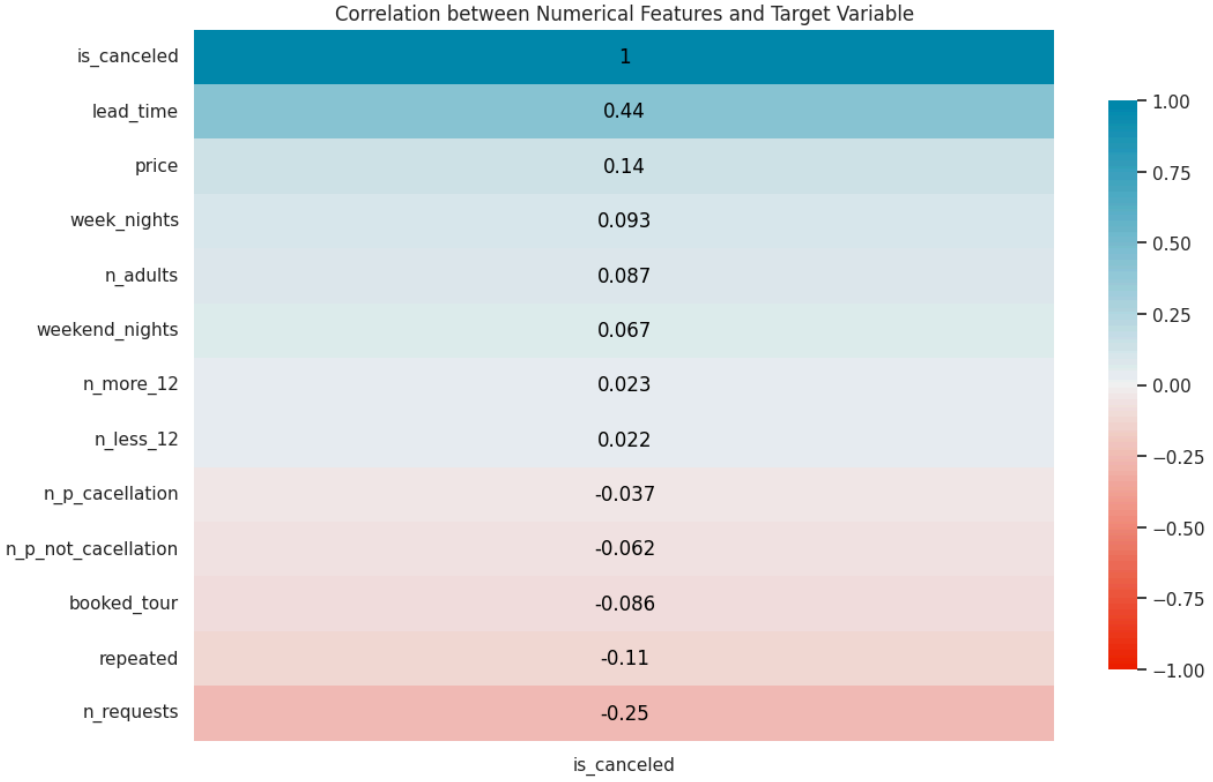
ככל שהערכים בטבלה יותר קרובים למינוס 1 או ל 1 הם יותר מתואמים.
ניתן לראות כי כל ערך מתואם עם עצמו ב100% (מן הסתם) ומוצג בצבע אדום בוהק תחת הערך -1.

ניתן לראות שיש קשר בין החישוב של האם הלקוח הוא לקוח חוזר להאם בערך היו לאותו הלקוח הזמנות והוא ביט אותן או שמה לא ביטל . (כיוון שאם הוא הזמין בעבר Repeated אזי בהכרח הוא ביטל או לא ביטל הזמנה קודמת)

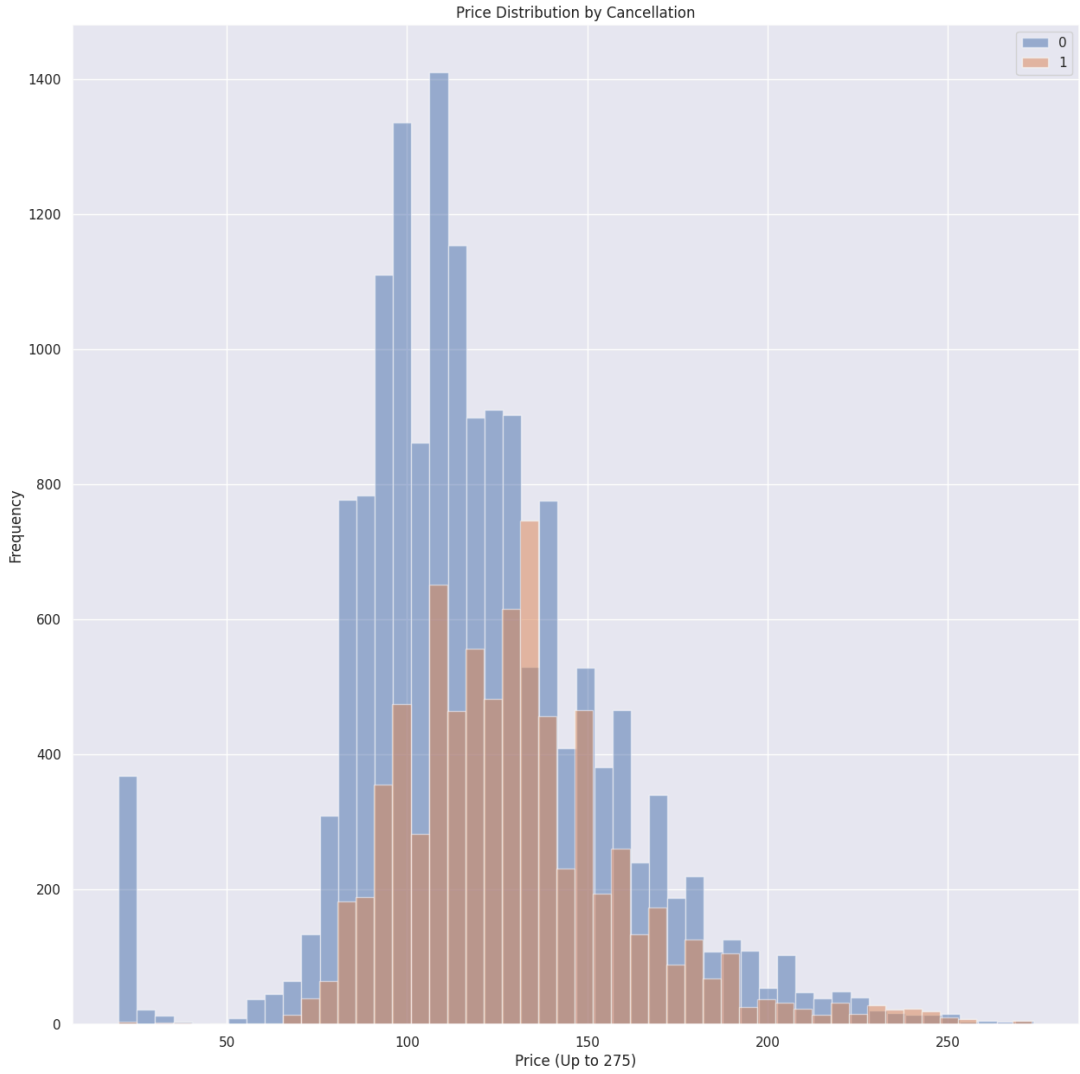
כיוון שאין מספר עמודות מרובה ונראה חוץ מהמקרה המצויין למעלה שאין כפילות בדאטה וכן כיוון שחישוב ומציאת הערכים הריקים והשלמתם עבור העמודות בעלות הערכים החסרים נעשה בדרך אחרת ויותר פשוטה.
ערכים חשובים שבדקנו במקרה שלנו באמצעות ההצגה הטבלאית הזו על מנת להבין טוב יותר את הטבלה
הינם הקורצליות בין העמודות השונות לערך המטרה בשורה האחרונה

שם ניתן לראות את חשיבותבין היתר של ה-price, number of requests, lead time כעמודות חשובות ביחס לפרדיקציה שאנו נדרשים לה האם הלקוח יבטל או לא יבטל את ההזמנה שלו.
כעמודות שערכן רחוק בטבלה מערך ה-0.

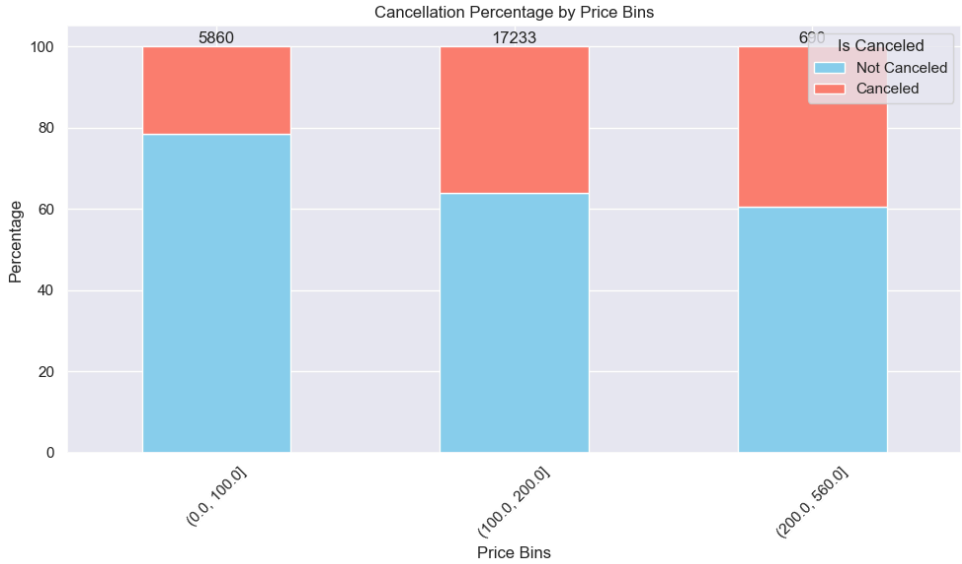
כיוון שכך יחדנו לכך ויזואלזציה נוספת:

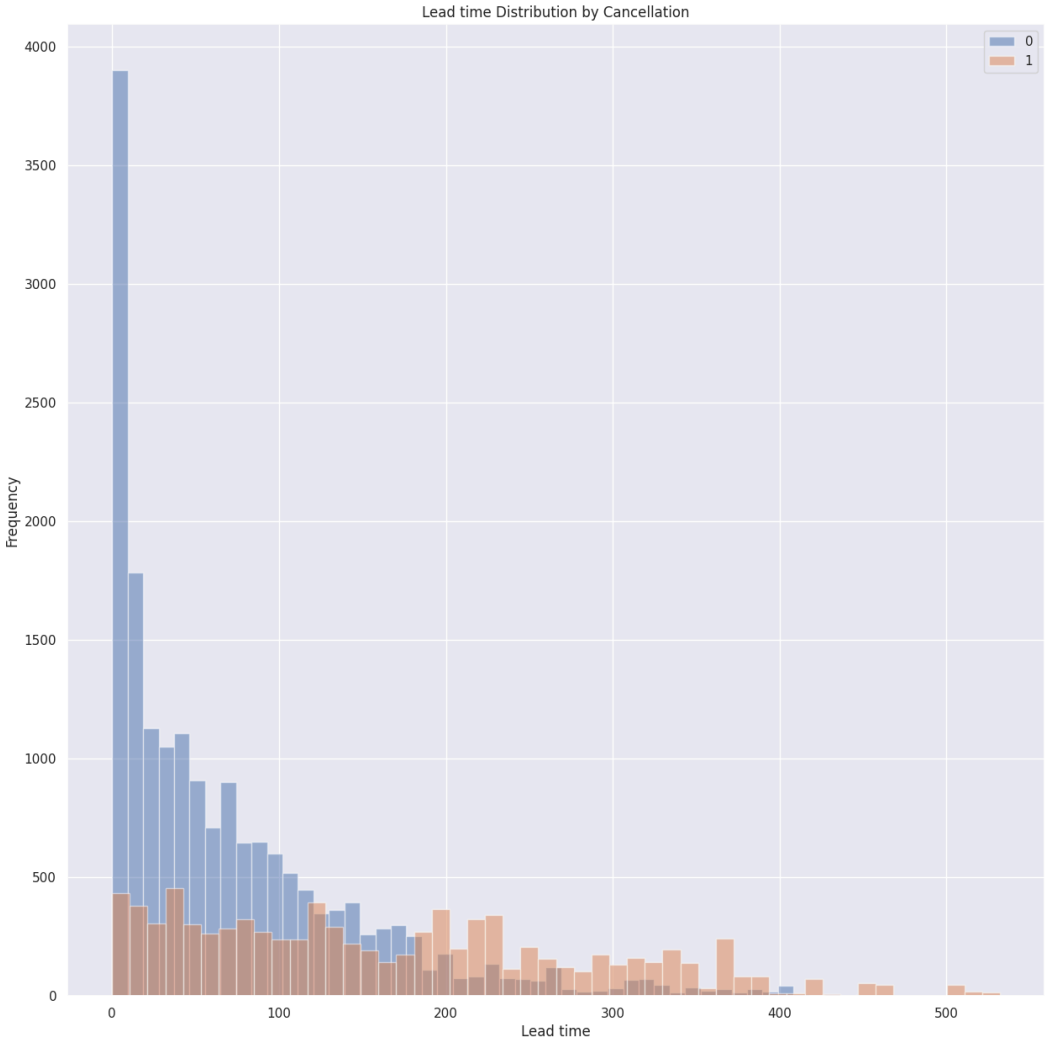


כיוון שמדובר בערכים מספריים רציפים בחרנו להציג את **Lead time** ואת **Price** גם בהיסטוגרמה רציפה המחולקת למקטעים של **10** , המציגה באדום את כמות הערכים בטווח הנתון שעבורם בוטלה ההזמנה(**1**) .
לעומת ההיסטוגרמות בצבע הכחול שמציינת כמו ערכים בטווח עבורם ההזמנה לא בוטלה (**0**) .



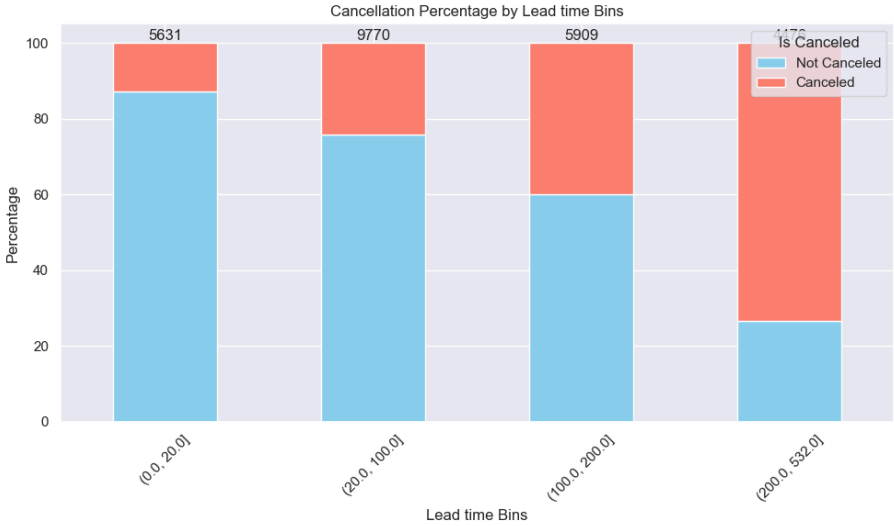
ניתן לראות שישנה הטייה מסוימת עבור ביטולים כתלות במחיר, מחיר גבוה גורר יחס גבוה יותר של ביטולים
נראה זאת גם באמצעות הגרף הבא. בוא הצגנו זאת ב3 גרפי עמודה המחולקים ל**BINS** - טווחי מחיר





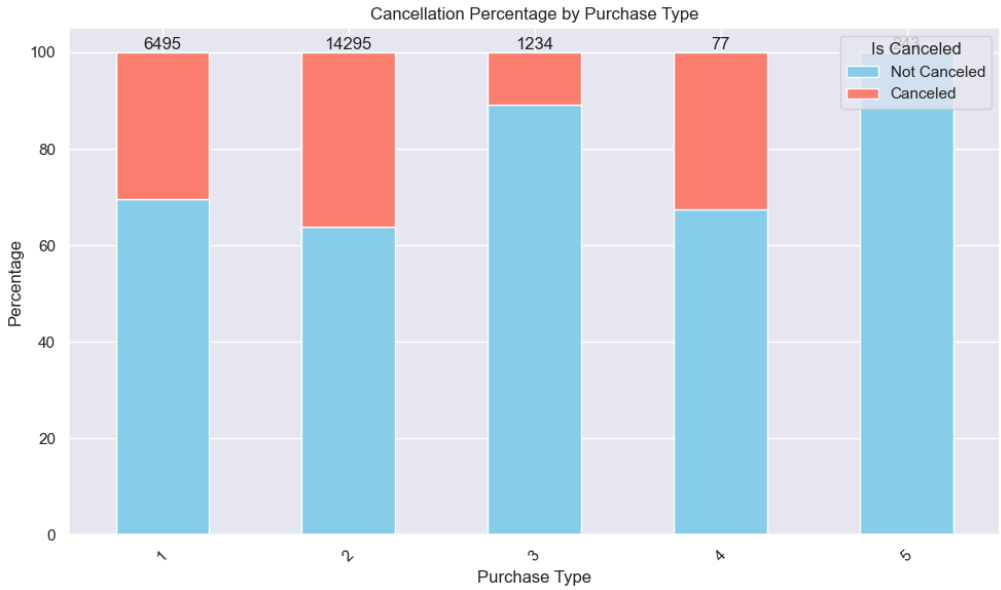
בנוסף ניתן לראות כאן את מה שראינו בטבלת הקורלציה - על הקשר החזק בין הזמן מביצוע ההזמנה לתאריך שנשמר.
כך שעבור זמני המתנה קצרים אחוז הביטולים קטן יחסית בעוד ככל שזמן ההמתנה גדל באופן כללי,והיחס אפילו מתהפך כך שכמות הביטולים גדולה מכמות אי- הביטולים !(מאיזור 200 ימים והלאה).

גם כאן ההצגה ב bins עבור הLead time , עוזרת לראות במפורש את שינוי המגמה ביחס הביטולים כתלות בזמן ההזמנה מראש.



את ההצגה ההיסטוגרמית לעיל ביצענו גם עבור העמודות בעלות הערכים הקטגורים ביחס לסיכוי לבטל באחוזים

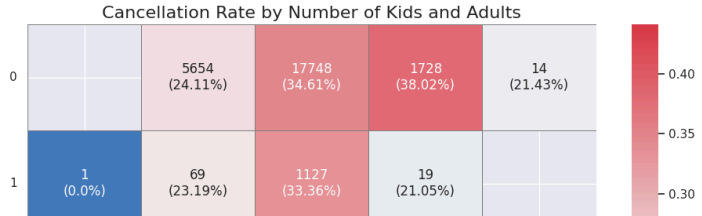
נציג לדוגמא את Purchase type . בהן ניתן לראות כי עבור הזמנות מסוג 3,5 ישנו אחוז ביטולים קטן בצורה משמעותית משאר סוגי ההזמנות מה שמאפשר להסיק שידיעת סוג ההזמנה עבור הזמנה עתידית תאפשר לחזות בצורה טובה יותר האם היא תבוטל או לא ולכן ישנה חשיבות שניקח אותו בהמשך בחשבון ובבניית מודלי החיזוי שלנו כנ"ל עבור שאר העמודות שעבורן ביצענו את ההשוואה.



בנוסף ערכנו גם טבלת השוואה עבור 3 משתנים, בטבלה זו המצורפת לעיל ניתן לראות השילוב בין כמות המבוגרים לילדים בהזמנה, ואת הסיכוי שלביטול ההזמנה עבור כל אחד מהשילובים, אחוז הביטולים עבור הצירוף מוצג בסוגריים, ומספר המופעים עבור צירוף יחיד מופיע כמספר.

***<u>לדוגמא</u>:** עבור **0** ילדים ו**2** מבוגרים אחוז הביטולים עומד על כ **34.61%** בעוד כמות הפעמים שמופיע בדאטה האימון הזמנה שמכילה **2** מבוגרים בדאטה הינה כ**17,778** .*

כאן ניתן לראות שעבור שילובים מסויימים של כמו ילדים ומבוגרים שמופיעים בצורה שכיחה דיה בדאטה שלנו יש שוני בין הממוצע ביטולים הכולל של כלל הרשומות לחלק מקומבינציות (נשים לב כי מספר הקומבינציות הינו משמעותי כאש ישנן מספר לא מבוטל של קומבינציות זניחות עקב מספר נמוך ביותר של מופעים בדאטה שלנו)



Data Preprocessing

- כהכנה ולקראת ההטמעה במודלי הלמידת מכונה עלינו לנרמל את הנתונים בעמודות המספריות ביצענו זאת באמצעות אובייקט **MinMaxScaler** אותו יבאנו מספריית sklearn עבור preprocessing.

- את העמודות הקטגוריות שהעברנו לייצוג מספרי במקום שמי , הגדרנו כINT.

Data balancing למה זה חשוב?

חוסר איזון בכמות הדוגמאות (Class Imbalance):
חוסר איזון בכמות הדוגמאות בין המעמדות השונים יכול להוביל למודל שאינו מתפקד היטב על המעמד המיעוטי. לדוגמה, אם רק אחוז קטן מההזמנות במלון מבוטלות, מודל שלא מתמודד עם חוסר איזון עלול לנבא שהזמנות כמעט ולא מתבטלות, מה שלא נכון בפועל.

שימוש ב-Borderline SMOTE:
טכניקת Borderline SMOTE מייצרת דוגמאות חדשות למעמד המיעוט, בעיקר באזור הגבול (boundary) של המעמד. זה מאפשר למודל ללמוד טוב יותר את התכונות הייחודיות של המעמד המיעוטי ולשפר את ביצועי המודל על נתונים לא מאוזנים.

ערבוב הנתונים:
ערבוב הנתונים מבטיח שהנתונים מפוזרים באופן אקראי, מה שמונע הטיות במהלך האימון ומבטיח שהמודל יקבל תמונה נכונה ומלאה של הנתונים.

חלוקה למערכי אימון ובדיקה:
חלוקה נכונה למערכי אימון ובדיקה מאפשרת להעריך את ביצועי המודל על נתונים שלא נראו במהלך האימון, מה שנותן אינדיקציה טובה יותר לביצועי המודל על נתונים חדשים ואמיתיים.

במקרה שלנו חלוקת היחס בין האימון לבדיקה הינו 80% -20%
כאשר ערך המטרה היינו ה -is canceled .

```python
X = train_df.drop('is_canceled', axis=1)  # Features
y = train_df['is_canceled']  # Target

smote = BorderlineSMOTE() # Initialize the BorderlineSMOTE resampler
X_resampled, y_resampled = smote.fit_resample(X, y) # Apply Borderline

df_resampled = pd.DataFrame(X_resampled, columns=X.columns)
df_resampled['is_canceled'] = y_resampled

df_resampled_shuffled = df_resampled.sample(frac=1, random_state=42)
```

```python
Y_resampled = df_resampled_shuffled['is_canceled']
X_resampled = df_resampled_shuffled.drop('is_canceled', axis=1)
```

```python
labels = Y_resampled
training_data = X_resampled
```

```python
print((labels == 1).sum())
print((labels == 0).sum())
```

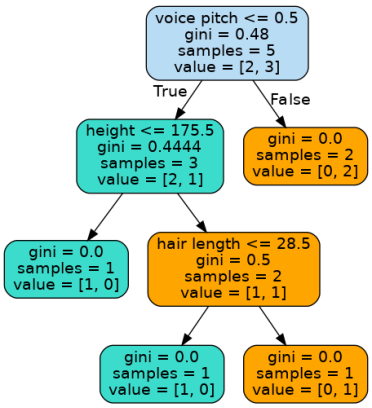
```python
x_train , x_test , y_train , y_test = train_test_split(training_data, labels , test_size = 0.2)
```

Data Models and Hyperparameters Explanations

המודלים בהם השתמשנו:

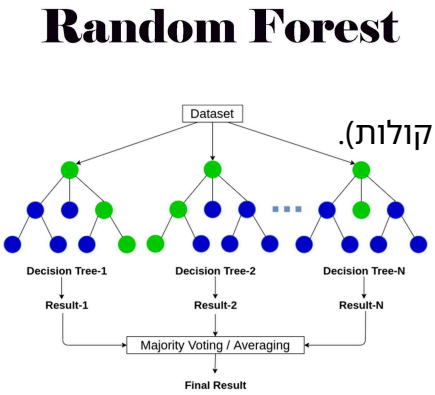
Decision Tree

(עץ החלטות) הוא מודל חיזוי שבו הנתונים מחולקים לצמתים על פי קריטריונים מסוימים
(כמו gini או entropy).
כל צומת בעץ מייצג פיצול של הנתונים למספר תתי קבוצות,
עד שמגיעים לעלי העץ המייצגים את התחזיות הסופיות.
מתחילים משורש העץ ובוחרים את הפיצול הראשון על בסיס קריטריון כלשהו
(למשל, gini או entropy).חוזרים על התהליך עבור כל תת קבוצה שנוצרת
עד שמגיעים לעלי העץ או לעומק מקסימלי מוגדר מראש.



<u>יתרונות:</u>
קל להבנה ולפירוש.
דורש הכנה מינימלית של הנתונים.
מתאים לבעיות סיווג (classification)

Random Forest (יער אקראי) הוא מודל המורכב ממספר רב של עצי החלטה.
המודל משתמש בריבוי עצים כדי לשפר את הדיוק ולמנוע overfitting. התחזיות של כל עצי ההחלטה
משולבות יחד כדי לתת תחזית סופית.
בונים מספר רב של עצי החלטה על תתי-קבוצות שונות של הנתונים.
משלבים את התחזיות של כל העצים כדי לקבל תחזית סופית (למשל, ממוצע או רוב קולות).



<u>יתרונות:</u>
מספק ביצועים גבוהים בהשוואה לעץ החלטות בודד.
פחות נוטה ל-overfitting.

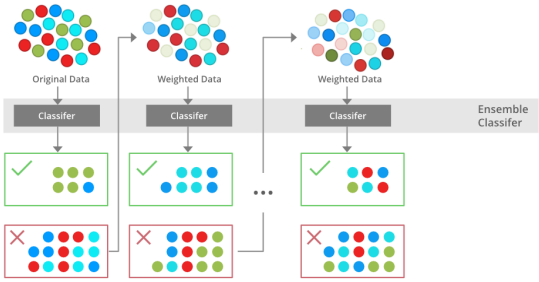
מתאים לטיפול בבעיות סיווג ורגרסיה.

Adaptive Boosting(AdaBoost) הוא אלגוריתם Boosting שמשתמש במספר מודלים חלשים (לרוב עצי החלטה קטנים) כדי ליצור מודל חזק. האלגוריתם מתאים משקלים גבוהים יותר לדוגמאות שהמודלים הקודמים נכשלו בחיזוי שלהן.

מתחילים עם מודל חלש ומאמן אותו על כל הנתונים.

מודלים נוספים מאומנים על דוגמאות שהמודלים הקודמים נכשלו בחיזוי שלהן.

כל המודלים משולבים יחד כדי ליצור תחזית סופית.



<u>יתרונות:</u>

משפר את הביצועים של מודלים חלשים.

מתאים לבעיות סיווג ורגרסיה.

Extreme Gradient Boosting(XGBoost) הוא אלגוריתם Boosting מתקדם שמשתמש בעצים כמו AdaBoost, אך משפר את הביצועים באמצעות טכניקות שונות כמו קיצוץ (pruning) וניהול זיכרון יעיל.

בונים עצי החלטה באופן איטרטיבי, כאשר כל עץ משפר את התחזיות של העצים הקודמים.

משתמשים במידע מהשגיאות של העצים הקודמים כדי לשפר את התחזיות הבאות.

<u>יתרונות:</u>

מספק ביצועים גבוהים מאוד.

מתאים לטיפול בבעיות סיווג ורגרסיה.

כולל אופטימיזציות רבות שמשפרות את הביצועים ומהירות האימון.

Model hyperparameters meaning and range explanation

הסבר על הטווחים וההיפרפרמטרים שנבחרו לכל אחד מהמודלים

1. Decision Tree

היפרפרמטרים:

`**criterion**`: הקריטריון המשמש לפיצול הצמתים בעץ. הערכים האפשריים הם 'gini' ו-'entropy'.

`**max_depth**`: העומק המקסימלי של העץ. ערכים אפשריים הם מספרים שלמים שונים ו-None (ללא הגבלה).

`**max_leaf_nodes**`: מספר העלים המרבי בעץ. ערכים אפשריים הם מספרים שלמים שונים ו-None.

`**min_samples_split**`: המספר המינימלי של דוגמאות הדרוש לפיצול צומת פנימי. ערכים אפשריים הם מספרים שלמים שונים ומספרים עשרוניים בין 0 ל-1.

`**min_samples_leaf**`: המספר המינימלי של דוגמאות הדרושות להיות בעלה. ערכים אפשריים הם מספרים שלמים שונים.

טווחים שנבחרו:

```
criterion = ['gini', 'entropy']
max_depth = [10, 20, 60, 70, 80, 90, 100, None]
max_leaf_nodes = [2, 5, 20, 30, 40, 50, None]
min_samples_split = [0.1, 0.5, 1.0, 2, 3, 4, 5]
min_samples_leaf = [1, 2, 3, 4, 5]
```

הסבר לטווח:

`**criterion**`: שתי האפשרויות הנפוצות ביותר לפיצול עצים, מה שמאפשר למודל לבדוק איזה קריטריון מספק ביצועים טובים יותר.

`**max_depth**`: טווח רחב של ערכים כדי לבדוק את השפעת עומק העץ על הביצועים ולמנוע overfitting.
`**max_leaf_nodes**`: בחירת ערכים שונים של מספר עלים כדי לבדוק את השפעתם על הביצועים.
`**min_samples_split**` ו-`**min_samples_leaf**`: טווחים שונים כדי לבדוק את השפעת מספר הדוגמאות המינימלי לפיצול ועלה על הביצועים. ערכים נמוכים מדי יכולים להוביל ל-overfitting וערכים גבוהים מדי יכולים להוביל ל-underfitting.

2. Random Forest

היפרפרמטרים:
`**n_estimators**`: מספר העצים ביער.
`**criterion**`: הקריטריון המשמש לפיצול הצמתים בעץ.
`**max_depth**`: העומק המקסימלי של כל עץ.
`**min_samples_split**`: המספר המינימלי של דוגמאות הדרוש לפיצול צומת פנימי.
`**min_samples_leaf**`: המספר המינימלי של דוגמאות הדרושות להיות בעלה.
`**max_features**`: מספר התכונות המרבי להיחשב לפיצול.

טווחים שנבחרו:

```
random_parameters = {'criterion' : ['gini', 'entropy'],
                     'n_estimators' : [50, 60, 70, 80, 90 ,100],
                     'max_depth' : [4 ,5, 6, 7, 8, None],
                     'min_samples_leaf': [0.1, 0.5, 1.0, 2, 5, 10, 15],
                     'min_samples_split' : [0.1, 0.5, 1.0 , 2, 5, 10, 15],
                     'max_features' : ['auto', 'log2']}
```

היגיון מאחורי הבחירה:
`**n_estimators**`: בחירת מספרים שונים של עצים כדי לבדוק את השפעת כמות העצים על הביצועים ולמצוא את האיזון בין דיוק זמן חישוב.
שאר ההיפרפרמטרים דומים לאלו של Decision Tree, אך מותאמים ל-Random Forest כדי לבדוק את השפעתם על יער של עצים רבים.
`**max_features**`: בחירת מספר התכונות המרבי כדי לבדוק איך שינוי במספר התכונות משפיע על הפיצול ועל ביצועי המודל.

3. AdaBoost

היפרפרמטרים:
`**n_estimators**`: מספר המודלים החלשים (weak learners).
`**learning_rate**`: קצב הלמידה שמכוון את תרומת כל מודל חלש למודל הסופי.

טווחים שנבחרו:

```
learning_rate = [0.1*(i+1) for i in range(15)]
n_estimators = [20 * (i + 20) for i in range(10)]
```

היגיון מאחורי הבחירה:

`**n_estimators**`: בחירת מספרים שונים של מודלים חלשים כדי לבדוק את השפעת כמות המודלים על הביצועים.

`**learning_rate**`: בחירת ערכים שונים של קצב הלמידה כדי לבדוק את השפעתם על הביצועים. קצב למידה נמוך מדי עלול לגרום למודל לא ללמוד מספיק, בעוד שקצב למידה גבוה מדי עלול לגרום ל-overfitting.

XGBoost .4

היפרפרמטרים:

`**n_estimators**`: מספר הסיבובים של boosting.

`**learning_rate**`: קצב הלמידה שמקטין את תרומת כל עץ כדי למנוע overfitting.

`**max_depth**`: העומק המקסימלי של כל עץ.

`**min_child_weight**`: הסכום המינימלי של משקלי הדוגמאות הדרוש בצומת.

`**subsample**`: אחוז הדוגמאות שמשתמשים בהן לכל עץ.

`**colsample_bytree**`: אחוז התכונות שמשתמשים בהן לכל עץ.

`**gamma**`: ההפחתה המינימלית בהפסד הנדרשת לפיצול צומת.

טווחים שנבחרו:

```
random_parameters = {
    'n_estimators': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
    'max_depth': [4, 5, 6, 7, 8, 9, 10],
    'learning_rate': [0.01, 0.05, 0.1, 0.2, 0.3],
    'subsample': [0.6, 0.7, 0.8, 0.9, 1.0],
    'colsample_bytree': [0.6, 0.7, 0.8, 0.9, 1.0],
    'gamma': [0, 0.1, 0.2, 0.3, 0.4, 0.5]
```

היגיון מאחורי הבחירה:

`**n_estimators**`: בחירת מספרים שונים של סיבובים כדי לבדוק את השפעת כמות הסיבובים על הביצועים ולמצוא את האיזון בין דיוק זמן חישוב.

`**max_depth**`: בחירת ערכים שונים של עומק העץ כדי לבדוק את השפעת עומק העץ על הביצועים ולמנוע overfitting.

`**learning_rate**`: בחירת ערכים שונים של קצב הלמידה כדי לבדוק את השפעתם על הביצועים.

`**min_child_weight**`: בחירת ערכים שונים כדי לבדוק את השפעת המשקלים המינימליים על הביצועים.
`**subsample**`: בחירת אחוזים שונים של הדוגמאות כדי לבדוק איך שינוי בכמות הדוגמאות שמשתמשים בהן משפיע על הביצועים.
`**colsample_bytree**`: בחירת אחוזים שונים של התכונות כדי לבדוק איך שינוי בכמות התכונות שמשתמשים בהן משפיע על הפיצול ועל ביצועי המודל.
`**gamma**`: בחירת ערכים שונים של הפחתת ההפסד כדי לבדוק את השפעתם על הפיצולים.

Models hyperparameters optimization

ביצוע אופטימיזציה לפרמטרי המודלים אנו בחרנו ב RandomizedSearchCV
למה צריך לבצע אופטימיזציה באמצעות **RandomizedSearchCV** על המודלים הללו?

כאשר אנחנו מאמנים מודל למידת מכונה, ישנם פרמטרים שנקבעים מראש, שנקראים היפרפרמטרים. היפרפרמטרים אלו משפיעים באופן משמעותי על ביצועי המודל. כיוון היפרפרמטרים (Hyperparameter Tuning) הוא תהליך חשוב שנועד למצוא את הערכים הטובים ביותר עבור היפרפרמטרים אלו כדי לשפר את ביצועי המודל.

הסיבות העיקריות לביצוע אופטימיזציה של המודלים באמצעות RandomizedSearchCV הן:

שיפור ביצועים:
בחירת ההיפרפרמטרים הנכונים יכולה לשפר באופן משמעותי את הדיוק, היציבות והיעילות של המודל.
מודלים עם היפרפרמטרים מכוונים היטב יכולים להשיג תוצאות טובות יותר בבדיקות ובתחזיות.
מניעת Overfitting ו-Underfitting:

אופטימיזציה של היפרפרמטרים יכולה לעזור למצוא את האיזון הנכון בין מורכבות המודל ופשטותו. מודל מורכב מדי עלול לבצע Overfitting וללמוד את רעשי הנתונים במקום את המגמות האמיתיות. מודל פשוט מדי עלול לבצע Underfitting ולא ללמוד מספיק את הנתונים.

שיפור הכללת המודל:

אופטימיזציה עוזרת למודל ללמוד בצורה טובה יותר ממערך הנתונים וליישם את הידע הזה על מערכי נתונים חדשים שלא נראו במהלך האימון, ובכך לשפר את היכולת של המודל להכליל (generalize) על נתונים חדשים.

חסכון בזמן ובמשאבים:

RandomizedSearchCV מאפשר לבדוק מספר רב של שילובים של היפרפרמטרים בצורה יעילה ומהירה. במקום לבדוק את כל השילובים האפשריים, RandomizedSearchCV בודק מספר קבוע של שילובים אקראיים, מה שמאפשר להגיע לתוצאות טובות בפחות זמן חישוב.

RandomizedSearchCV מאפשר לנו לבדוק מספר רב של שילובים של היפרפרמטרים באופן אקראי מתוך טווח מוגדר מראש. זה נותן לנו מספר יתרונות:

יעילות בזמן חישוב:

RandomizedSearchCV בודק מספר קבוע של שילובים אקראיים זה חשוב במיוחד כשיש לנו הרבה היפרפרמטרים וטווחים רחבים של ערכים.

יכולת לכסות טווח רחב של ערכים:

מכיוון ש-RandomizedSearchCV בודק שילובים אקראיים, יש סיכוי גבוה יותר למצוא שילובים טובים של היפרפרמטרים מבלי לבדוק את כל השילובים האפשריים.

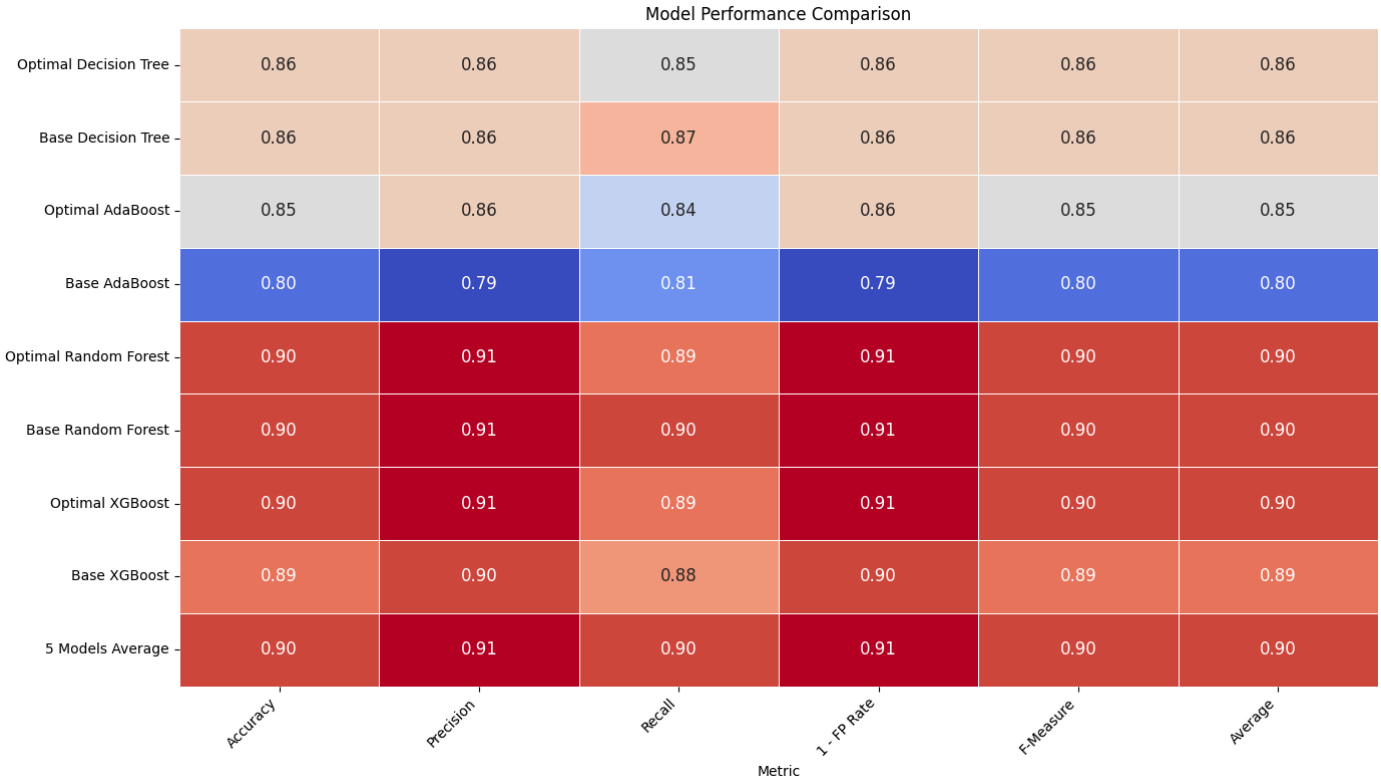
שיפור ביצועים:

RandomizedSearchCV מאפשר למצוא שילובים טובים של היפרפרמטרים שמשפרים את ביצועי המודל מבלי לבצע חיפוש ממצה של כל השילובים האפשריים, מה שיכול להיות בלתי מעשי כשיש הרבה היפרפרמטרים.

Model Training and preformence Evaluation

עבור פרויקט חיזוי ביטול הזמנות בבתי מלון, נבחרו מודלים שונים של למידת מכונה כדי להעריך את הביצועים שלהם בחיזוי ביטולים.

עבור כל אחד מהמודלים חישבנו accuracy , presition,recall,fpr,f-score. בהתאם לערכי ה tp,tn,fp,fn שמצאנו:



סיכום תוצאות המודלים בטבלה כאשר ניתן לראות בצד שמאל בציר ה-Y את המודלים השונים המכילים גרסה לפני אופטימיזציה ואחריה

ובציר ה X ניתן לראות את התוצאות ערכי המטרה השונים , כאשר בעמודה הימנית מחושב הממוצע שלהם
*מכאן אנו רואים כי עלינו לבחור את שני מודלים ה-**Random forest** , שני מודלי ה- **XGBoost** ומודל חמישי ה**Base** **decision Tree** עבור כל רשומה נבצע פרדיקציה עבור כל אחד מהמודלים ונחליט האם ערך המטרה '**Is_Canceled**' שווה אפס או אחד על פי הרוב*
(הרוב מ5, ז"א 3 ומעלה)

Test preprocessing and prediction

עבור בחינת הקובץ ה-test

נשמור את הקובץ כdata frame באמצעות pandas כ - test_df
נבצע שינויים בדאטה כמו שיבצענו עבור קובץ הtrain
ניתן ערך מספרי למידע קטגורי room_type,board_type,purchase_type
נשלים ערכים חסרים בדומה לדרך שפורטה עבור ה-train
נוריד את עמודת ה-ID
ננרמל את הנתונים המספריים בין 0 ל-1.
עבור העמודות הקטגוריות שהצגנו במספרים נציג אותן במשתנים מטיפוס -int.

שמרנו את הדאטה לאחר הכנה

הרצנו על 5 המודלים וביצענו עבורו חישוב שבו נלך אחרי הרב:
נחלק ב 3 ללא שארית ככה שעבור חיזוי של P=1 עבור 3 מודלים ומעלה נחזה P=1 אחרת נחזה עבור אותה רשומה P=0

```
p1, p2, p3, p4, p5 = rf.predict(test_df), rf_random.predict(test_df), xgb_random.predict(test_df), dtc.predict(test_df), xgb.predict(test_df)
p = (p1 + p2 + p3 + p4 + p5)//3
p = list(p)
#add predictions for invalid dates
for pos in invalid_date_indices:
    p.insert(pos, 0)
```

נוסיף את העמודה is_canceled למודל ונשמור אותו כ-test_with_predictions .

(*כלל קטעי הקוד מופיעים במחברת הפייתון וcsv עם התוצאות מצורף בקובץ הזיפ.)