

Hotel Cancellation Analysis

Iftehzaz Newaz

Introduction

Hotel businesses face challenges in maximizing occupancy rates and income due to cancellations, which can result in increased staffing and operational costs, empty rooms, and lost revenue. Therefore, it is crucial for hotel owners to understand the causes of cancellations and devise strategies to reduce them. Questions such as the reasons behind cancellations, the type of analysis to be conducted, and the most important features to consider often arise in this context.

This report discusses the analysis of Hotel Booking data for two hotels in Portugal using PySpark, a machine learning tool, with visualization done in Tableau.

The aim of this report is to develop a robust and accurate model for predicting hotel booking cancellations by considering a wide range of relevant factors. The datasets were collected with the goal of developing prediction models to classify the likelihood of a hotel booking being canceled. However, these datasets can also be utilized beyond the scope of cancellation prediction due to the characteristics of the variables included.

In this project, I have focused on predicting whether a hotel booking will be canceled or not, making it a classification problem. I have considered various algorithms, such as Logistic Regression, Decision Tree, Random Forest, and Gradient Boosting Tree, for predicting cancellations.

Dataset [Link](#)

In end, I attempt to solve those questions. This is my perspective of the outcome, it's based on the dataset itself. The current world scenario might be different from it.

Related Work

The problem of hotel reservation booking cancellation has been widely studied in the hospitality industry, given its significant impact on the industry's revenue and operations. Researchers have used various techniques, such as machine learning and statistical models, to predict hotel reservation booking cancellations and improve hotel management.

One study by Gao et al. (2020) used a deep learning approach to predict hotel reservation cancellations. They applied a Long Short-Term Memory (LSTM) network to a dataset of hotel reservations and achieved a high accuracy rate of 87% in predicting cancellations. Similarly, another study by Alzahrani et al. (2021) used an ensemble learning approach to predict cancellations. They combined decision trees, random forests, and support vector machines to create a model that achieved an accuracy rate of 80.23%.

Researchers have also explored statistical models, such as logistic regression, to predict hotel reservation cancellations. A study by Fernandes et al. (2020) used a logistic regression model to identify the factors that influence hotel reservation cancellations. They found that factors such as the lead time, type of booking, and rate type significantly affect cancellations.

The aforementioned studies underscore the significance of predicting hotel reservation cancellations and demonstrate how machine learning and statistical models can enhance hotel management. Despite the effectiveness of these models, accurately predicting hotel reservation cancellations still presents challenges, especially regarding the limitations of the available data and the high dimensionality of the problem. Therefore, there is still a need for improvement in the prediction accuracy of hotel reservation cancellations.

Implementation

➤ Apache Spark

The phrase "Big Data" has been regularly used in recent months by significant technological companies. Big Data may be identified by its size, processing power, effectiveness, and—critically—the fact that information frequently comes in an unstructured manner, making it expensive and difficult to manage. This is where Apache Spark steps in, operating as a distributed memory system that maintains data on top of the Hadoop File System (HDFS), providing more stability than Hadoop MapReduce. In conclusion, Apache Spark is an open-source Big Data processing solution that offers a variety of capabilities to enable SQL queries, machine learning, and continuous data streaming inside a single system. Apache Spark Language Support:

- Apache Spark (Java and Scala Support)
- **PySpark (Python Support)**
- SparkR (R support)

Note, this report only contains information about PySpark.

➤ **PySpark:**

The Python programming language's PySpark library supports all of Spark's functionality, including SQL, streaming, and, most importantly, machine learning via the Pandas API. PySpark provides the interface between the Spark framework and Python. Similar to how Python facilitates machine learning and complex data analysis, it also provides interactive compatibility with Jupyter. In this investigation, we'll examine JupyterLab's PySpark installation procedure. However, it is important to note that PySpark's support is mostly restricted to the Pandas API and does not include additional libraries, such as NumPy, which is a drawback.

Also, I used PySpark in the project of Hotel Booking Cancellation analysis with machine learning algorithm and preprocessing.

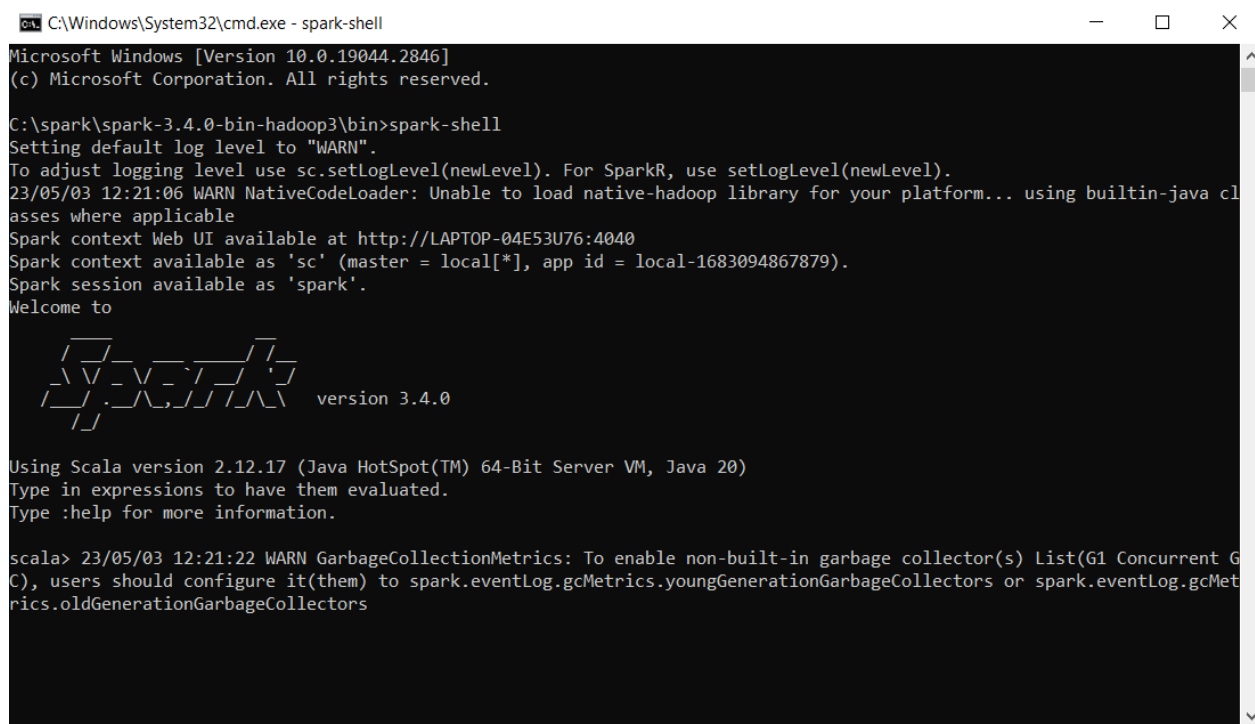
➤ Install PySpark in Jupyter Notebook

I have been using Jupyter Notebook for the past two years for machine learning as well as deep learning.

So, I have used Jupyter Notebook as my working platform. It's easy to use and easy to install packages.

- First, we have to install JDK 20 [LINK](#)
- Secondly, Download Spark [LINK](#)
- Finally, Install Python Latest version [LINK](#)

Install everything on 'C' Drive, Set environment variables according to directory above softwares are installed. Go to the command prompt and type "spark-shell", your spark connection will be established.



```
C:\Windows\System32\cmd.exe - spark-shell
Microsoft Windows [Version 10.0.19044.2846]
(c) Microsoft Corporation. All rights reserved.

C:\spark\spark-3.4.0-bin-hadoop3\bin>spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/05/03 12:21:06 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Spark context Web UI available at http://LAPTOP-04E53U76:4040
Spark context available as 'sc' (master = local[*], app id = local-1683094867879).
Spark session available as 'spark'.
Welcome to

  ____
 /    \
/_  _/
 \_  _/
  /_/

 version 3.4.0

Using Scala version 2.12.17 (Java HotSpot(TM) 64-Bit Server VM, Java 20)
Type in expressions to have them evaluated.
Type :help for more information.

scala> 23/05/03 12:21:22 WARN GarbageCollectionMetrics: To enable non-built-in garbage collector(s) List(G1 Concurrent GC), users should configure it(them) to spark.eventLog.gcMetrics.youngGenerationGarbageCollectors or spark.eventLog.gcMetrics.oldGenerationGarbageCollectors
```

Fig 1.1: Installing pyspark

Fortunately I did not get any errors.

- **Dataset:**

The dataset for this project, sourced from Kaggle, includes historical information on hotel reservations for two hotels in Portugal, a city hotel (H2) and a resort hotel (H1), during the years 2015 to 2017. The dataset consists of 32 variables and 119,390 observations. Some of the key variables include hotel type, booking cancellation status, lead time, arrival date, length of stay (weekend and week nights), and number of adults. The target variable, "is_canceled," indicates whether or not a reservation was canceled.

ADR: Average Daily Rate.

Adults: Number of Adults.

Agent: ID of the travel agency that made the booking.

ArrivalDateDayOfMonth: Day of the month of the arrival date.

ArrivalDateMonth: Month of arrival date with 12 categories: "January" to "December".

ArrivalDateWeekNumber: Week number of the arrival date.

ArrivalDateYear: Year of arrival date.

AssignedRoomType: Code for the type of room assigned to the booking. Sometimes the assigned room type differs from the reserved room type due to hotel operation reasons (e.g. overbooking) or by customer request. Code is presented instead of designation for anonymity reasons.

Babies: Number of babies.

BookingChanges: Number of changes/amendments made to the booking from the moment the booking was entered on the PMS until the moment of check-in or cancellation.

Children: Number of children.

Company: ID of the company/entity that made the booking or responsible for paying the booking. ID is presented instead of designation for anonymity reasons.

Country: Country of origin..

CustomerType: type of booking, assuming one of four categories

Contract - when the booking has an allotment or other type of contract associated to it;

Group – when the booking is associated to a group;

Transient – when the booking is not part of a group or contract, and is not associated to other transient booking;

Transient-party – when the booking is transient, but is associated to at least other transient booking.

DaysInWaitingList: Number of days the booking was in the waiting list before it was confirmed to the customer.

DepositType: Indication on if the customer made a deposit to guarantee the booking..

DistributionChannel: Booking distribution channel. The term “TA” means “Travel Agents” and “TO” means “Tour Operators”.

IsCanceled: Value indicating if the booking was canceled (1) or not (0).

IsRepeatedGuest: Value indicating if the booking name was from a repeated guest (1) or not (0).

LeadTime: Number of days that elapsed between the entering date of the booking into the PMS and the arrival date.

MarketSegment: Market segment designation. In categories, the term “TA” means “Travel Agents” and “TO” means “Tour Operators”.

Meal: Type of meal booked.

PreviousBookingsNotCanceled: Number of previous bookings not cancelled by the customer prior to the current booking.

PreviousCancellations: Number of previous bookings that were cancelled by the customer prior to the current booking.

RequiredCardParkingSpaces: Number of car parking spaces required by the customer.

ReservationStatus: Reservation last status: Canceled – booking was canceled by the customer;

Check-Out – customer has checked in but already departed;

No-Show – customer did not check-in and did inform the hotel of the reason why.

ReservationStatusDate: Date at which the last status was set. This variable can be used in conjunction with the ReservationStatus to understand when was the booking canceled or when did the customer checked-out of the hotel.

ReservedRoomType: Code of room type reserved. Code is presented instead of designation for anonymity reasons.

StaysInWeekendNights: Number of weekend nights (Saturday or Sunday) the guest stayed or booked to stay at the hotel.

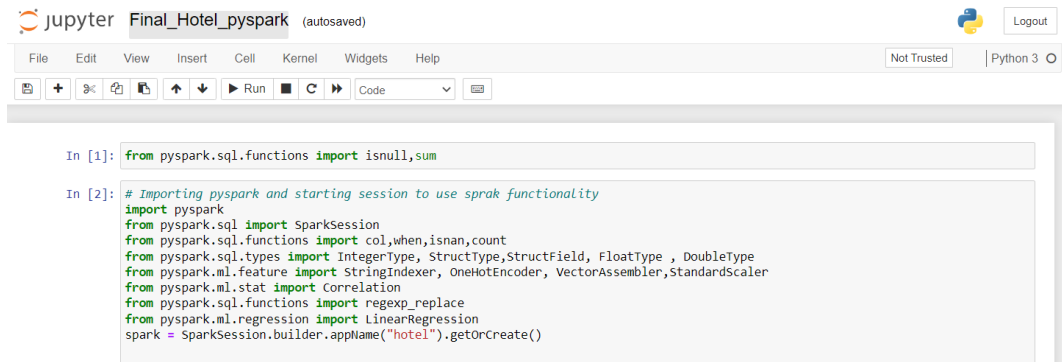
StaysInWeekNights: Number of week nights (Monday to Friday) the guest stayed or booked to stay at the hotel.

TotalOfSpecialRequests: Number of special requests made by the customer (e.g. twin bed or high floor)

I covered all the attributes/columns here.

- **Evaluation and Machine Learning approaches:**

At first, I imported the necessary library. Then, I created a SparkSession known as “hotel”. With PySpark and SQL, Machine learning model works better with full efficiency. Here is a screenshot of my code.

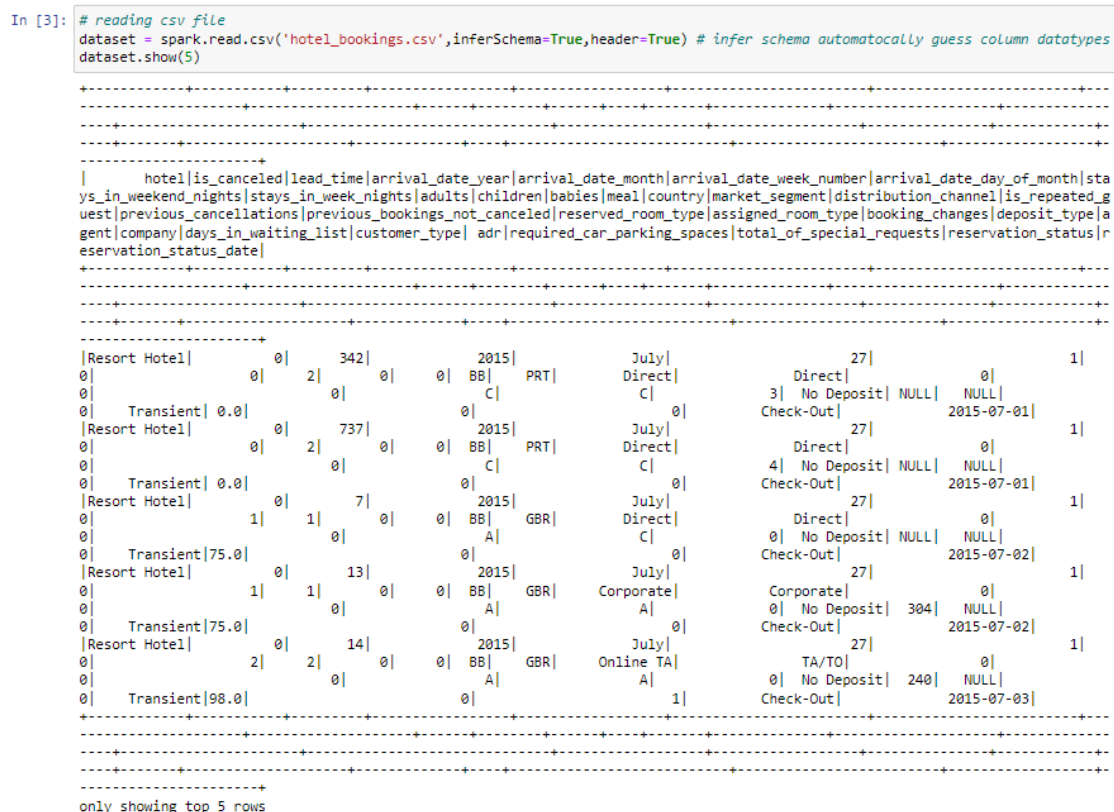


```
In [1]: from pyspark.sql.functions import isnull, sum

In [2]: # Importing pyspark and starting session to use sprak functionality
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when, isnan, count
from pyspark.sql.types import IntegerType, StructType, StructField, FloatType, DoubleType
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler, StandardScaler
from pyspark.ml.stat import Correlation
from pyspark.sql.functions import regexp_replace
from pyspark.ml.regression import LinearRegression
spark = SparkSession.builder.appName("hotel").getOrCreate()
```

Fig 1.2 Importing library

I have kept the dataset on the same directory where the notebook file is saved.



```
In [3]: # reading csv file
dataset = spark.read.csv('hotel_bookings.csv', inferSchema=True, header=True) # infer schema automotacolly guess column datatypes
dataset.show(5)
```

is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies	meal	country	market_segment	distribution_channel	is_repeated_guest	previous_cancellations	previous_bookings_not_canceled	reserved_room_type	assigned_room_type	booking_changes	deposit_type	agent	company	days_in_waiting_list	customer_type	adr	required_car_parking_spaces	total_of_special_requests	reservation_status	reservation_status_date
0	0	2015	July	27	1	0	2	342	0	0	BB	PRT	Direct	Direct	0	0	0	BB	C	3	No Deposit	NULL	NULL	0	0	0	0	0	0	2015-07-01
0	0	2015	July	27	1	0	2	737	0	0	BB	PRT	Direct	Direct	0	0	0	BB	C	4	No Deposit	NULL	NULL	0	0	0	0	0	0	2015-07-01
0	0	2015	July	27	1	0	1	7	0	0	BB	GBR	Direct	Direct	0	0	0	BB	A	0	No Deposit	NULL	NULL	0	0	0	0	0	0	2015-07-02
0	0	2015	July	27	1	0	1	13	0	0	BB	GBR	Corporate	Corporate	0	0	0	BB	A	0	No Deposit	304	NULL	0	0	0	0	0	0	2015-07-02
0	0	2015	July	27	1	0	2	14	0	0	BB	GBR	Online TA	TA/TO	0	0	0	BB	A	0	No Deposit	240	NULL	0	0	0	0	0	0	2015-07-03

only showing top 5 rows

Fig 1.3 Loading Dataset


```
In [4]: dataset.printSchema()

root
|-- hotel: string (nullable = true)
|-- is_canceled: integer (nullable = true)
|-- lead_time: integer (nullable = true)
|-- arrival_date_year: integer (nullable = true)
|-- arrival_date_month: string (nullable = true)
|-- arrival_date_week_number: integer (nullable = true)
|-- arrival_date_day_of_month: integer (nullable = true)
|-- stays_in_weekend_nights: integer (nullable = true)
|-- stays_in_week_nights: integer (nullable = true)
|-- adults: integer (nullable = true)
|-- children: string (nullable = true)
|-- babies: integer (nullable = true)
|-- meal: string (nullable = true)
|-- country: string (nullable = true)
|-- market_segment: string (nullable = true)
|-- distribution_channel: string (nullable = true)
|-- is_repeated_guest: integer (nullable = true)
|-- previous_cancellations: integer (nullable = true)
|-- previous_bookings_not_canceled: integer (nullable = true)
|-- reserved_room_type: string (nullable = true)
|-- assigned_room_type: string (nullable = true)
|-- booking_changes: integer (nullable = true)
|-- deposit_type: string (nullable = true)
|-- agent: string (nullable = true)
|-- company: string (nullable = true)
|-- days_in_waiting_list: integer (nullable = true)
|-- customer_type: string (nullable = true)
|-- adr: double (nullable = true)
|-- required_car_parking_spaces: integer (nullable = true)
|-- total_of_special_requests: integer (nullable = true)
|-- reservation_status: string (nullable = true)
|-- reservation_status_date: date (nullable = true)
```

Fig 1.4: Schema

After that, I printed the schema of the dataset like an SQL table. It contains columns name, and data types and checks whether the ISnullable or not.

```
In [5]: dataset.count()
Out[5]: 119390

In [6]: distinct_counts = {}

for col in dataset.columns:
    distinct_counts[col] = dataset.select(col).distinct().count()

print(distinct_counts)

{'hotel': 2, 'is_canceled': 2, 'lead_time': 479, 'arrival_date_year': 3, 'arrival_date_month': 12, 'arrival_date_week_number': 53, 'arrival_date_day_of_month': 31, 'stays_in_weekend_nights': 17, 'stays_in_week_nights': 35, 'adults': 14, 'children': 6, 'babies': 5, 'meal': 5, 'country': 178, 'market_segment': 8, 'distribution_channel': 5, 'is_repeated_guest': 2, 'previous_cancellations': 15, 'previous_bookings_not_canceled': 73, 'reserved_room_type': 10, 'assigned_room_type': 12, 'booking_changes': 21, 'deposit_type': 3, 'agent': 334, 'company': 353, 'days_in_waiting_list': 128, 'customer_type': 4, 'adr': 8879, 'required_car_parking_spaces': 5, 'total_of_special_requests': 6, 'reservation_status': 3, 'reservation_status_date': 926}
```

Fig 1.5 Checking for duplicate values

```

In [7]: from pyspark.sql.functions import count, countDistinct

# Create an empty dictionary to store the duplicate counts for each column
duplicate_counts = {}

# Loop over all columns in the DataFrame
for col in dataset.columns:
    # Count the total number of rows in the column
    total_count = dataset.select(col).count()
    # Count the number of distinct rows in the column
    distinct_count = dataset.select(col).distinct().count()
    # Count the number of duplicate rows in the column
    duplicate_count = total_count - distinct_count
    # Store the duplicate count for this column in the dictionary
    duplicate_counts[col] = duplicate_count

# Print the duplicate counts for all columns
print(duplicate_counts)

{'hotel': 119388, 'is_canceled': 119388, 'lead_time': 118911, 'arrival_date_year': 119387, 'arrival_date_month': 119378, 'arrival_date_week_number': 119337, 'arrival_date_day_of_month': 119359, 'stays_in_weekend_nights': 119373, 'stays_in_week_nights': 119355, 'adults': 119376, 'children': 119384, 'babies': 119385, 'meal': 119385, 'country': 119212, 'market_segment': 119382, 'distribution_channel': 119385, 'is_repeated_guest': 119388, 'previous_cancellations': 119375, 'previous_bookings_not_canceled': 119317, 'reserved_room_type': 119380, 'assigned_room_type': 119378, 'booking_changes': 119369, 'deposit_type': 119387, 'agent': 119056, 'company': 119037, 'days_in_waiting_list': 119262, 'customer_type': 119386, 'adr': 110511, 'required_car_parking_spaces': 119385, 'total_of_special_requests': 119384, 'reservation_status': 119387, 'reservation_status_date': 118464}

In [8]: print("Number of rows in the original DataFrame:", dataset.count())
dataset = dataset.distinct() # dropping a distinct/duplicates values
dataset.count()
print("Number of rows after dropping duplicates DataFrame:", dataset.count())

Number of rows in the original DataFrame: 119390
Number of rows after dropping duplicates DataFrame: 87396

```

Fig 1.5 Dropping duplicate values

After that to get rid of the duplicate rows I called a distinct function. The number of rows went from 11930 to 87396.

```
In [9]: null_count = dataset.select([sum(isnull(c).cast("int")).alias(c) for c in dataset.columns])
null_count.show()
```

hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies	meal	country	market_segment	distribution_channel	is_repeated_guest	previous_cancellations	previous_bookings_not_cancelled	reserved_room_type	assigned_room_type	booking_changes	deposit_type	agent	company	days_in_waiting_list	customer_type	adr	required_car_parking_spaces	total_of_special_requests	reservation_status	reservation_status_date
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Fig 1.5 Checking the null values

Here I have called the function to check the number of null values in every columns, as we can see there was not any null values in the dataset.

```
In [10]: df = dataset.drop("adr", "country", "distribution_channel", "deposit_type", "agent", "company", "arrival_date_year", "arrival_date_month", "arrival_date_day_of_month")
df.show()
```

Resort Hotel	0	9	2015	July	13
0	1	1	2	0	0
0	0	0	0	0	0
0	0	Transient	1	A	2

Fig 1.6: Dropping unnecessary columns

Here, I have dropped some unnecessary columns.

```
In [11]: df = df.withColumnRenamed("arrival_date_year", "arrival_year").withColumnRenamed("arrival_date_month", "arrival_month")
df.show(5)
```

Resort Hotel	0	9	2015	July	13
1	2	0	0	HB	Online TA
0	0	A	0	A	0
1	2	0	0	Check-Out	2015-07-15
4	2	0	0	BB	Online TA
0	0	D	0	0	0
0	0	0	0	Canceled	2015-06-01
1	2	2	0	BB	Online TA
0	0	C	0	0	0
1	1	0	0	Check-Out	2015-07-17
5	2	0	0	HB	Offline TA/TO
0	0	E	0	E	0
0	0	1	0	Canceled	2015-06-17
2	2	0	0	BB	Online TA
0	0	A	0	D	0
0	2	2	0	Check-Out	2015-08-11

only showing top 5 rows

Fig 1.6: Renaming long columns

Here I have renamed few long columns so that it becomes readable.

```
In [18]: from pyspark.sql.functions import *
from pyspark.sql.functions import year, month, dayofweek
df = df.withColumn('dayOfWeek', dayofweek(col('reservation_status_date')))
df = df.withColumn('month', month(col('reservation_status_date')))
df = df.withColumn('year', year(col('reservation_status_date')))
```

Fig 1.6: Reformatting the Date column

- **Visualization Of Clean Data:**

For visualization, we need to use Tableau

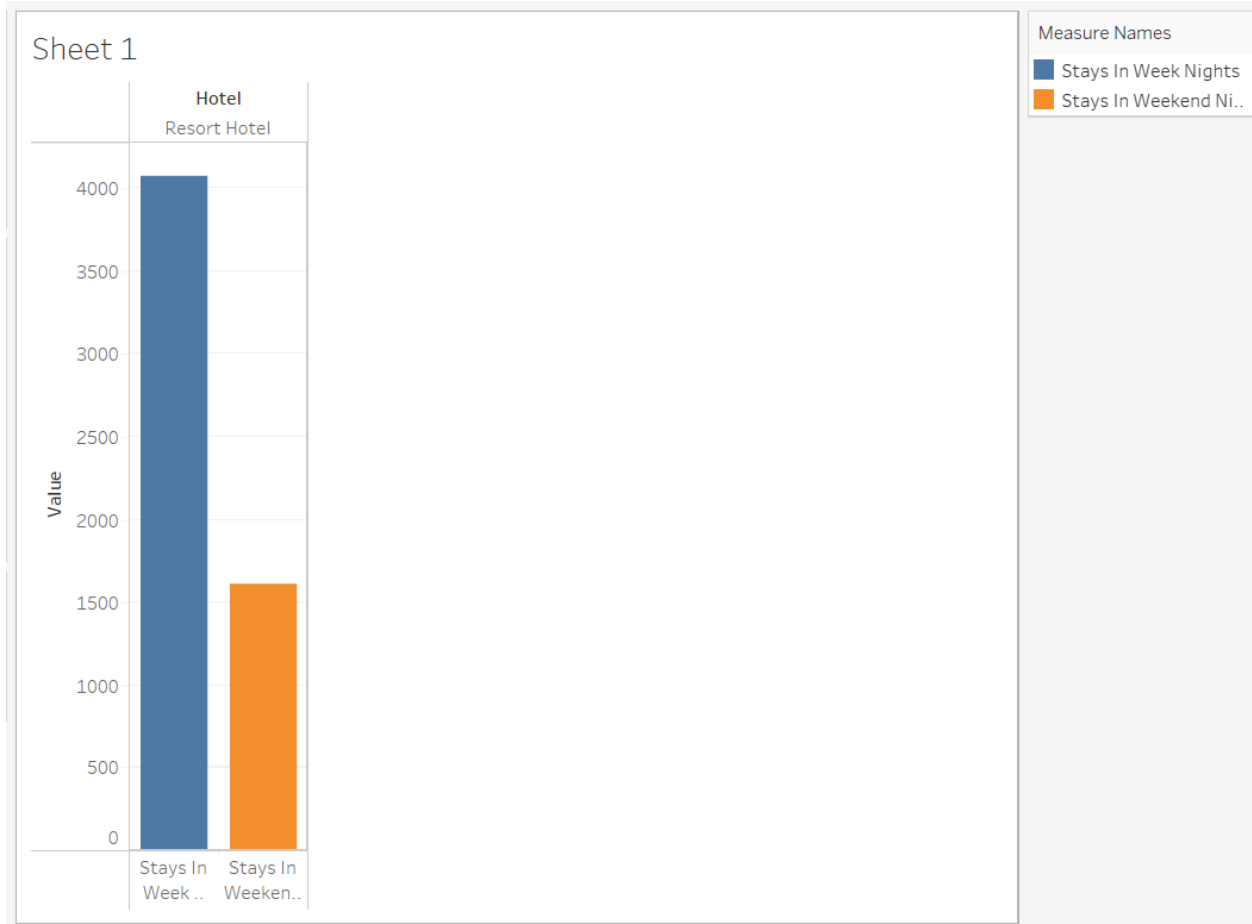


Fig 2.1 Bar-chart, number of bookings on weekend and weekdays.

This simple Bar-chart analysis shows the number of bookings take place on weekend and weekdays in resort.

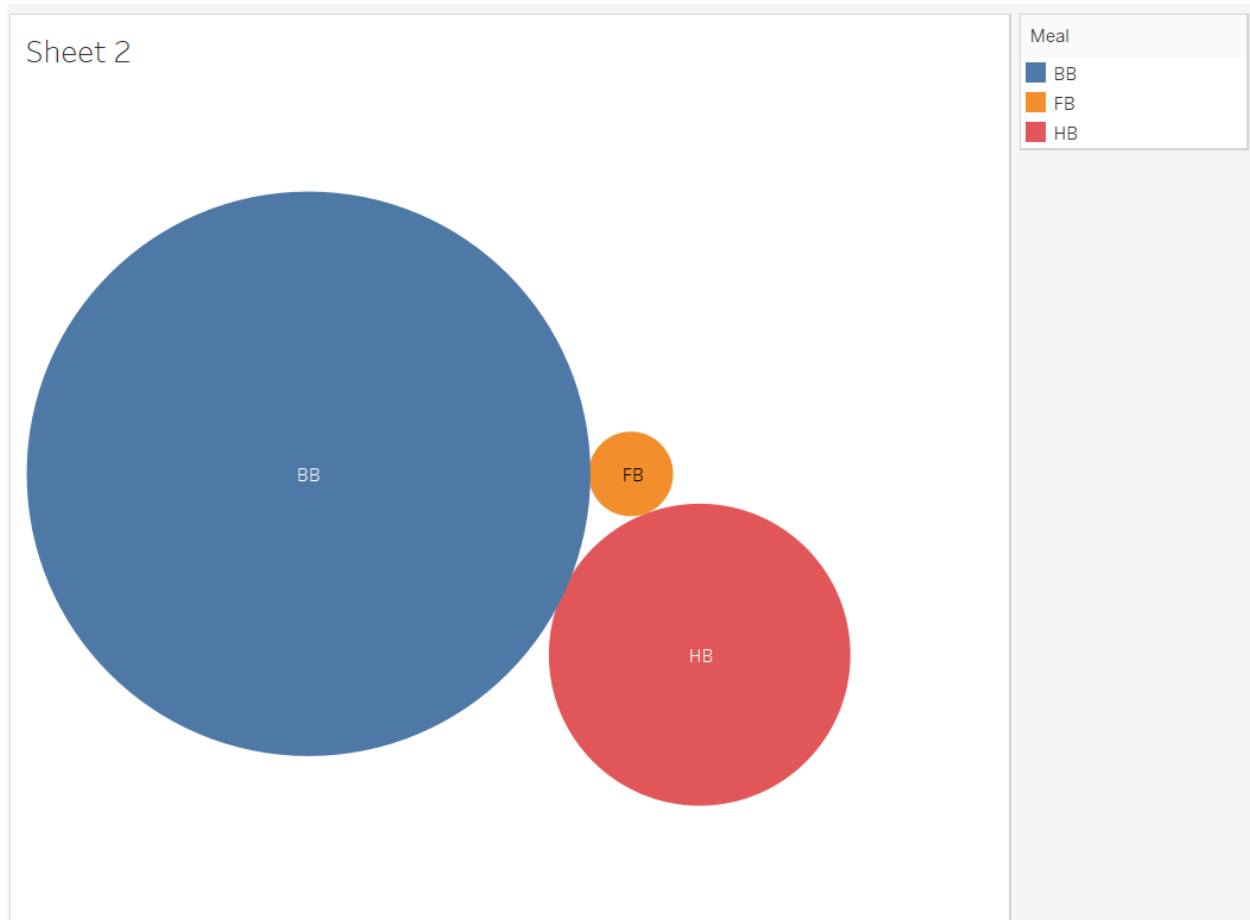


Fig 2.2 Type of meals tourist orders during their stay in the hotel.

This simple analysis shows the type of meals tourist orders during their stay in the hotel.

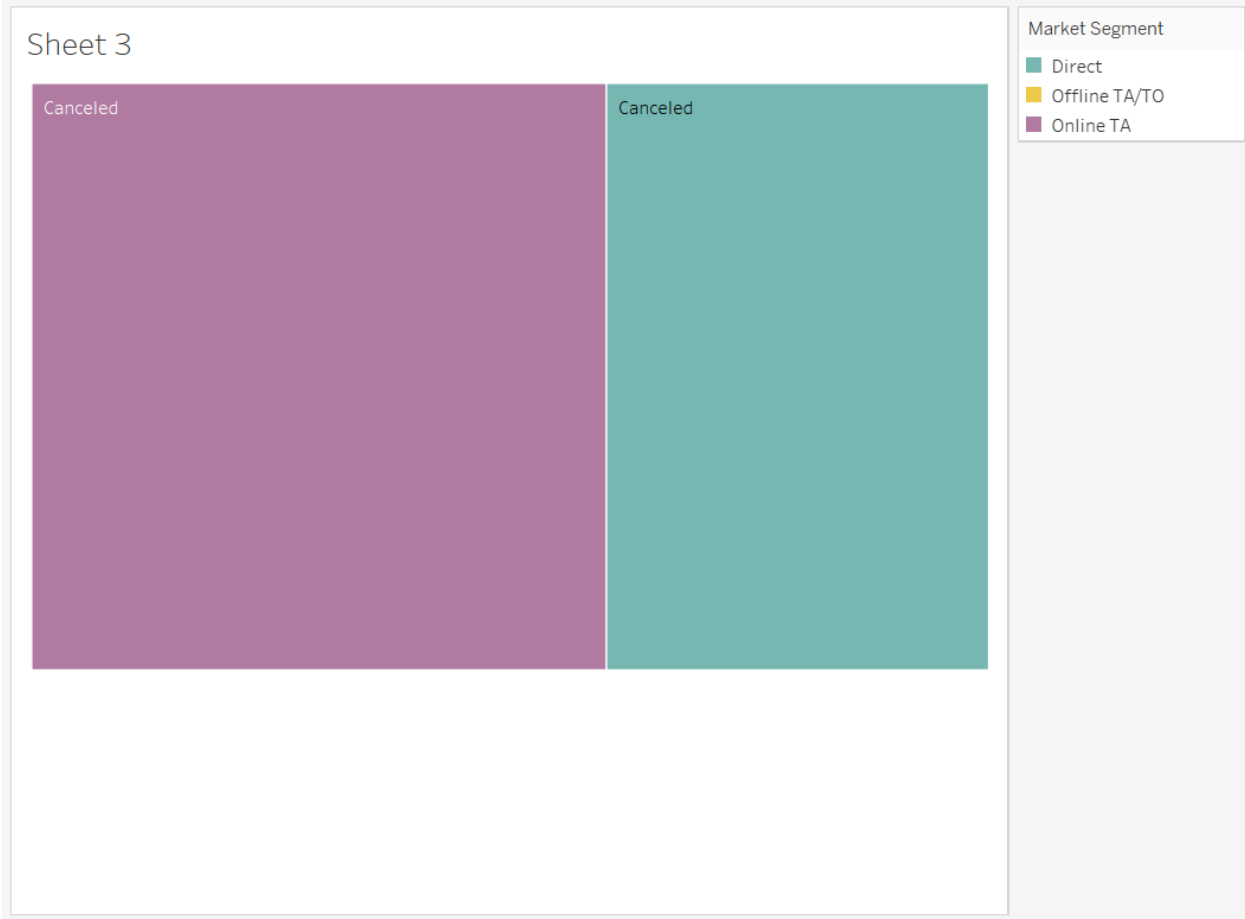


Fig 2.3 Reservation being cancelled based on market segment.

This simple analysis shows the amount of reservation being cancelled based on market segment.

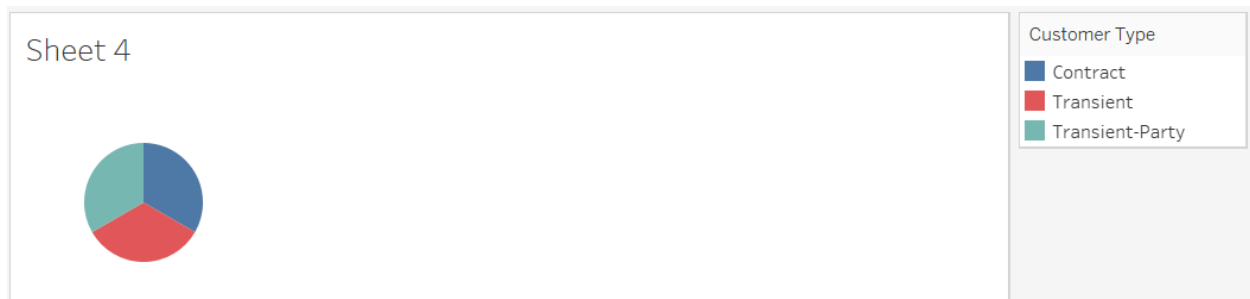


Fig 2.4 Tourist booking hotel based on the customer type.

This simple analysis shows the amount of booking reservation taking place based on the customer type.

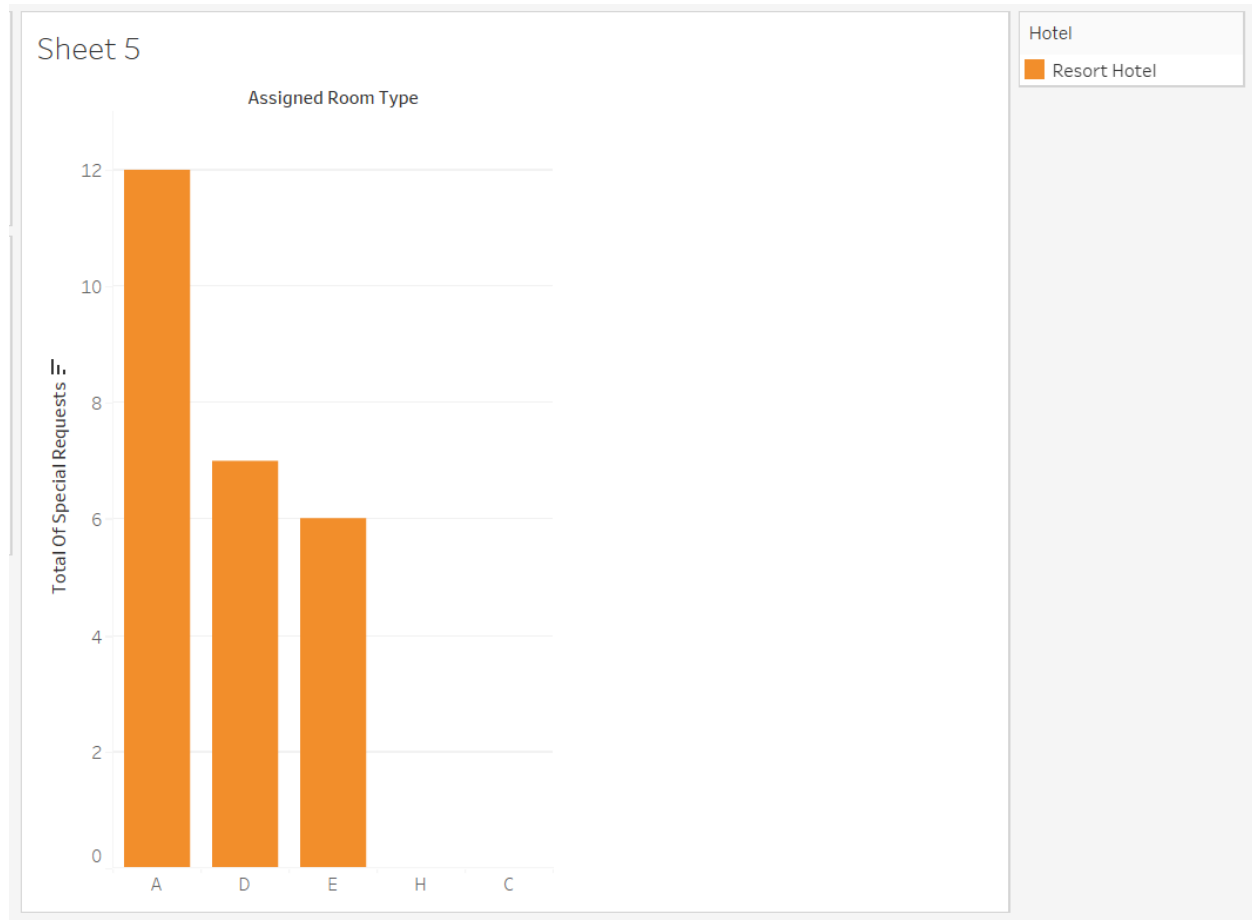


Fig 2.5 Amount of speacial request for particular room type.

This simple analysis shows the amount of speacial request for a particular room type.

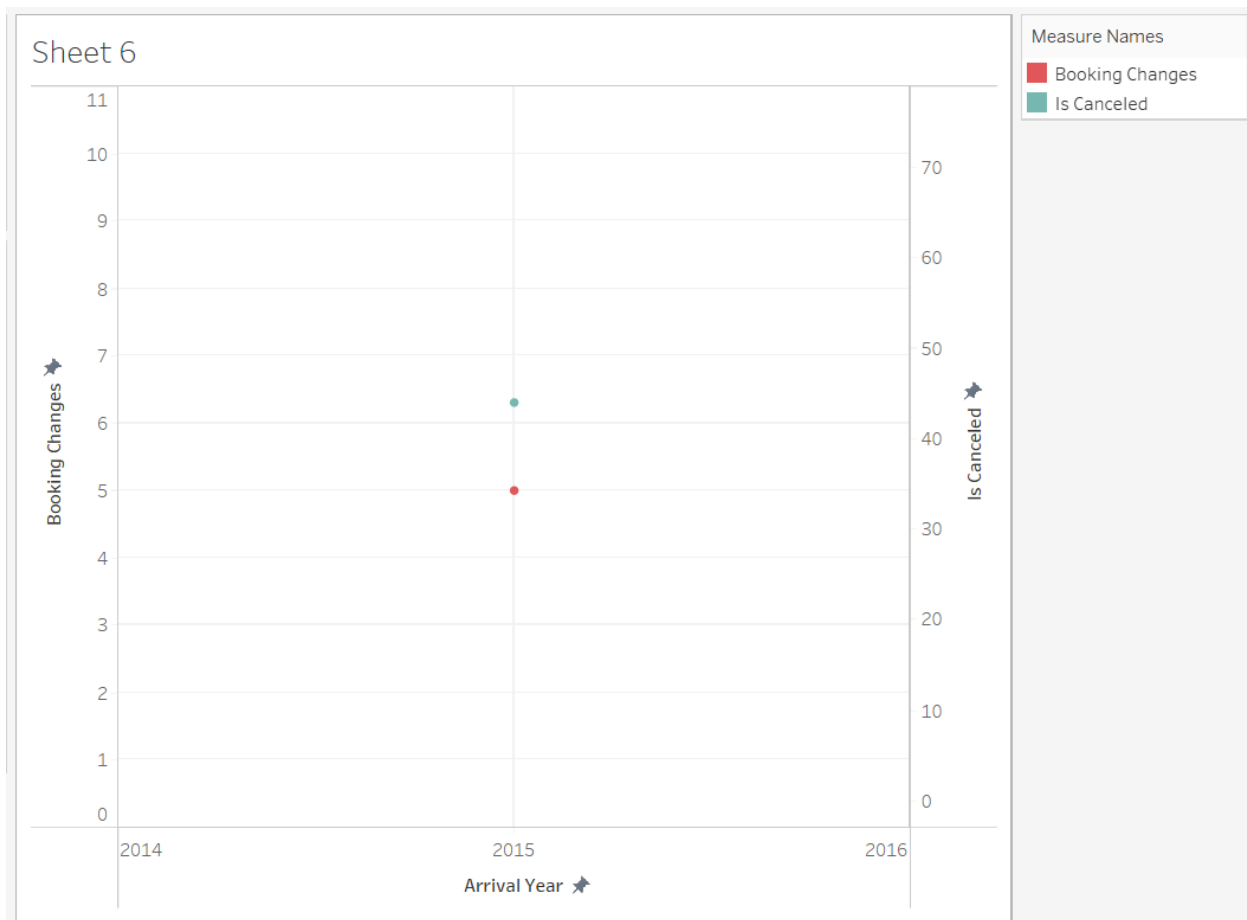


Fig 2.6 Amount of booking being canceled and changes in booking in a year.

This simple analysis shows the amount of booking being canceled and changes in booking in a particular year.

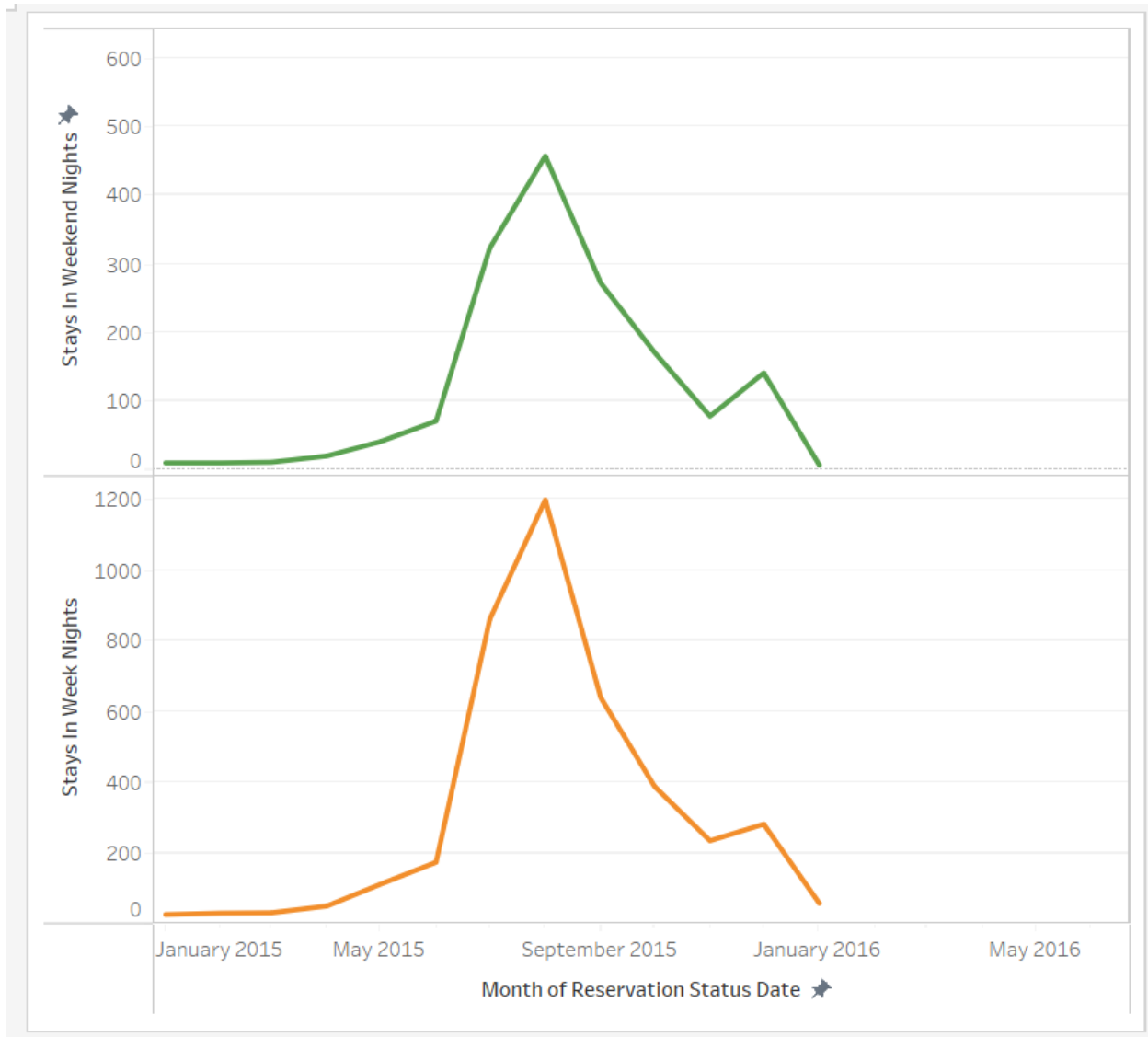


Fig 2.6 Amount of reservation taking place throughout the year.

This simple analysis shows the amount of reservation taking place throughout the year.

Experimental Section

To begin with, machine learning algorithms are based on mathematical concepts, and in the actual world, we have all forms of data such as strings, integers, doubles, and Booleans. It may occupy a large amount of machine memory and make machine learning less efficient. To address this issue, we use feature engineering to transform long strings to integers and a greater number of characteristics into a single vector known as the feature except label column. I utilised “StringIndexer” and “OneHotEncoding” in Pyspark.

```
In [15]: strinx = StringIndexer(inputCols = ["hotel","arrival_month","children","meal","market_segment","reserve  
In [16]: df = strinx.fit(df).transform(df)
```

Fig 3.1 StringIndexer

As the name implies, StringIndexer is typically used for categorising string data into indexes. It features a built-in mechanism that converts strings into indices and also casts the type of the string. Indexing begins at 0 and proceeds in descending order. Here we can witness string indexer's work on hotel, children, and many more.

```
In [20]: onenc = OneHotEncoder(inputCols = ["hotel_trans","arrival_month_trans","children_trans","meal_trans","n  
        outputCols = ["hotel_VEC","arrival_month_VEC","children_VEC","meal_VEC","market_se  
df = onenc.fit(df).transform(df)
```

```
In [21]: df.show(5,vertical=True)
```

```
-RECORD 0-----  
hotel          | Resort Hotel  
is_canceled    | 0  
lead_time      | 9  
arrival_year   | 2015  
arrival_month  | July  
arrival_day    | 13  
stays_in_weekend_nights | 1  
stays_in_week_nights  | 1  
adults         | 2  
children       | 0  
babies         | 0  
meal           | HB  
market_segment | Online TA  
is_repeated_guest | 0  
previous_cancellations | 0  
previous_bookings_not_canceled | 0  
reserved_room_type | A  
assigned_room_type | A
```

Fig 3.2 OneHotEncoder

A OneHotEncoder takes an index as input and outputs a binary vector. Because the last column is set to false by default, it may erase the last vector value.

As its name implies, `VectorAssembler` brings together all vectors into a single feature. In-depth, turn a string into indices first, then an index to a single hot encode. In the end, we will get a vector of categorical values, but now the algorithm needs all possible numerical values as well. In conclusion, we employ vector assembler.

Fig 3.3 VectorAssembler

MinMax Scaler:

As its name implies, Rescale each feature individually to a common range [min, max] linearly using column summary statistics, which is also known as min-max normalization or Rescaling.

```
In [26]: from pyspark.ml.feature import MinMaxScaler
# scaler = MinMaxScaler(inputCol="features", outputCol="features")
# scalerModel = scaler.fit(df_dropped)

scaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")
# rescale each feature to range [min, max].
scaledData = scaler.fit(df_dropped).transform(df_dropped)
scaledData.show()
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
|is_canceled|lead_time|arrival_year|arrival_day|stays_in_weekend_nights|stays_in_week_nights|adults|babies|is_repeated_guest|previous_cancellations|previous_bookings_not_canceled|booking_changes|days_in_waiting_list|required_car_parking_spaces|total_of_special_requests|hotel_trans|arrival_month|h_trans|children_trans|meal_trans|market_segment_trans|reserved_room_type_trans|assigned_room_type_trans|customer_type_trans|reservation_status_trans|dayOfWeek|month|year|hotel_VEC|arrival_month_VEC|children_VEC|meal_VEC|market_segment_VEC|reserved_room_type_VEC|assigned_room_type_VEC|customer_type_VEC|reservation_status_VEC|features|scaledFeatures|
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
```

Fig 3.3 MinMax Scaler

➤ **Machine Learning:**

Humans can learn through their surroundings, mistakes, and from other people. A machine can be programmed by humans to learn from its surroundings and data-driven behaviour. In essence, machine learning is the use of a machine that learns like a human and is used to perform complex calculations and resolve problems in the real world to improve human lives. AI includes machine learning as a subset.

There are three types of machine learning,

- **Supervised Learning:**

Labeled data is used in supervised learning to train models. In plain English, data with input and output will provide accuracy with test data based on the amount of training component. How precise our data really is.

- **Unsupervised Learning:**

It is completely at odds with guided learning. We only have inputs for this data, and the programmer doesn't even know what the output will be. As a result, machines themselves produce some output.

- **Reinforcement Learning:**

Environment and behavioral factors are important in reinforcement learning. Here, a machine uses trial and error to obtain a reward. Chess game one is the most well-known example.

For this dataset I have used Logistic Regression, Decision Tree, Random Forest and Gradient Boosting Tree algorithms.

➤ **Logistic Regression:**

The highest performance for a binary dependent variable is provided by logistic regression, which works best with predictive data. It enables the evaluation of numerous variable values with great accuracy.

➤ **Decision Tree:**

Decision tree learning is a supervised learning approach used in statistics, data mining and machine learning. In this formalism, a classification or regression decision tree is used as a predictive model to draw conclusions about a set of observations. Tree

models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels.

➤ **Random Forest:**

The random forest uses numerous decision trees to classify variables before arriving at a single result node. It can execute at various node levels and discover better results.

➤ **Gradient Boosting Tree:**

Gradient tree boosting is an ensemble learning method that used in regression and classification tasks in machine learning. The model improves the weak learners by different set of train data to improve the quality of fit and prediction..

➤ Machine Learning Algorithm Implementation:

Before applying machine learning algorithm, we have to split the data into test set and train set. Here, I have split the data 70% for training and 30% for testing.

```
In [27]: splitedData.randomSplit([0.7,0.3])
         train_df= splitedData[0]
         test_df = splitedData[1]
```

Fig 3.1 Splitting the data

After that, I applied machine learning algorithm.

Logistic Regression:

```
In [28]: #Logistic Regression
         from pyspark.ml.classification import LogisticRegression
         logic = LogisticRegression(featuresCol = 'features',labelCol = 'is_canceled', maxIter = 500)
         logicModel = logic.fit(train_df)
         predictions = logicModel.transform(test_df)
         predictions.select('is_canceled', 'rawPrediction', 'prediction', 'probability').show(10)
```

is_canceled	rawPrediction	prediction	probability
0	[19.6068810229503...	0.0	[0.99999999694620...
0	[19.7669960119323...	0.0	[0.99999999739802...
0	[20.0012668535961...	0.0	[0.99999999794145...
0	[19.7943833416078...	0.0	[0.99999999746832...
0	[19.5039269256352...	0.0	[0.99999999661505...
0	[19.8820073592314...	0.0	[0.99999999768071...
0	[19.4659800180892...	0.0	[0.99999999648413...
0	[19.0605170966538...	0.0	[0.99999999472621...
0	[19.4711047502582...	0.0	[0.99999999650210...
0	[19.0824349249650...	0.0	[0.99999999484054...

only showing top 10 rows

```
In [29]: from pyspark.ml.evaluation import BinaryClassificationEvaluator
         evaluator = BinaryClassificationEvaluator()
         evaluator.setLabelCol("is_canceled")
         evaluator.setRawPredictionCol("prediction")
         print('Test Area Under ROC', evaluator.evaluate(predictions))
```

Test Area Under ROC 1.0

Fig 3.1 Applying Logistic Regression

I have applied Logistic regression where maxiter stands for max iteration which is 500 and ROC of 1.0 in the test dataset. In other words, the model's predictions are close to the actual values.

Decision Tree:

Here, I have applied Decision Tree. Furthermore, maxDepth stands for maximum Depth for a tree which is set to 3.

```
In [31]: #Decision Tree
from pyspark.ml.classification import DecisionTreeClassifier
dt = DecisionTreeClassifier(featuresCol = 'scaledFeatures', labelCol = 'is_canceled', maxDepth = 3)
dtModel = dt.fit(train_df)
predictions = dtModel.transform(test_df)
predictions.select('is_canceled', 'rawPrediction', 'prediction', 'probability').show(10)
```

```
+-----+-----+-----+-----+
|is_canceled|rawPrediction|prediction|probability|
+-----+-----+-----+-----+
|          0|[44437.0,0.0]|        0.0| [1.0,0.0]|
|          0|[44437.0,0.0]|        0.0| [1.0,0.0]|
|          0|[44437.0,0.0]|        0.0| [1.0,0.0]|
|          0|[44437.0,0.0]|        0.0| [1.0,0.0]|
|          0|[44437.0,0.0]|        0.0| [1.0,0.0]|
|          0|[44437.0,0.0]|        0.0| [1.0,0.0]|
|          0|[44437.0,0.0]|        0.0| [1.0,0.0]|
|          0|[44437.0,0.0]|        0.0| [1.0,0.0]|
|          0|[44437.0,0.0]|        0.0| [1.0,0.0]|
|          0|[44437.0,0.0]|        0.0| [1.0,0.0]|
+-----+-----+-----+-----+
only showing top 10 rows
```

```
In [32]: from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator()
evaluator.setLabelCol("is_canceled")
evaluator.setRawPredictionCol("prediction")
print('Test Area Under ROC', evaluator.evaluate(predictions))
```

Test Area Under ROC 1.0

Fig 3.2 Decision Tree

Here, I have applied Decision Tree. Furthermore, we got ROC of 1.0 in the test dataset. In other words, the model's predictions are close to the actual values.

Random Forest Classifier:

Here, I have imported the required module and applied the algorithm.

```
In [33]: #Random Forrest
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(featuresCol = 'scaledFeatures', labelCol = 'is_canceled')
rfModel = rf.fit(train_df)
predictions = rfModel.transform(test_df)
predictions.select('is_canceled', 'rawPrediction', 'prediction', 'probability').show(10)
```

```
+-----+-----+-----+-----+
|is_canceled|rawPrediction|prediction|probability|
+-----+-----+-----+-----+
|          0|[44437.0,0.0]|         0.0|[1.0,0.0]|
|          0|[44437.0,0.0]|         0.0|[1.0,0.0]|
|          0|[44437.0,0.0]|         0.0|[1.0,0.0]|
|          0|[44437.0,0.0]|         0.0|[1.0,0.0]|
|          0|[44437.0,0.0]|         0.0|[1.0,0.0]|
|          0|[44437.0,0.0]|         0.0|[1.0,0.0]|
|          0|[44437.0,0.0]|         0.0|[1.0,0.0]|
|          0|[44437.0,0.0]|         0.0|[1.0,0.0]|
|          0|[44437.0,0.0]|         0.0|[1.0,0.0]|
|          0|[44437.0,0.0]|         0.0|[1.0,0.0]|
+-----+-----+-----+-----+
only showing top 10 rows
```

```
In [34]: from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator()
evaluator.setLabelCol("is_canceled")
evaluator.setRawPredictionCol("prediction")
print('Test Area Under ROC', evaluator.evaluate(predictions))

Test Area Under ROC 1.0
```

Fig 3.3 Applying Random Forest Classifier

Applying Random Forest I got ROC of 1.0 which is considered as a good result.

Gradient Boosting Tree:

```
In [35]: #Gradient Boosting
from pyspark.ml.classification import GBTClassifier
gbt = GBTClassifier(featuresCol = 'scaledFeatures', labelCol = 'is_canceled', maxIter=10)
gbtModel = gbt.fit(train_df)
predictions = gbtModel.transform(test_df)
predictions.select('is_canceled', 'rawPrediction', 'prediction', 'probability').show(10)
```

is_canceled	rawPrediction	prediction	probability
0	[1.32590267922034...	0.0	[0.93412217565278...
0	[1.32590267922034...	0.0	[0.93412217565278...
0	[1.32590267922034...	0.0	[0.93412217565278...
0	[1.32590267922034...	0.0	[0.93412217565278...
0	[1.32590267922034...	0.0	[0.93412217565278...
0	[1.32590267922034...	0.0	[0.93412217565278...
0	[1.32590267922034...	0.0	[0.93412217565278...
0	[1.32590267922034...	0.0	[0.93412217565278...
0	[1.32590267922034...	0.0	[0.93412217565278...
0	[1.32590267922034...	0.0	[0.93412217565278...

only showing top 10 rows

```
In [36]: from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator()
evaluator.setLabelCol("is_canceled")
evaluator.setRawPredictionCol("prediction")
print('Test Area Under ROC', evaluator.evaluate(predictions))
```

Test Area Under ROC 1.0

Fig 3.1 Applying Gradient Boosting Tree

I have applied Gradient Boosting Tree where maxiter stands for max iteration which is 10 and ROC of 1.0 in the test dataset. In other words, the model's predictions are close to the actual values.

Discussion of Findings

The study employed Logistic Regression and Random Forest Classifier among other classification models to forecast hotel booking cancellations. Using metrics such as ROC, the models' performance was evaluated on the testing set. The analysis of the dataset suggests that it is large enough to accurately predict hotel booking cancellations using only machine learning algorithms. The chosen algorithms demonstrated exceptional performance and achieved the project's objective. Therefore, the study's results indicate that classification models are highly effective in predicting hotel booking cancellations, which is beneficial for both hosts and guests.

Conclusion

We analyzed the data and created a classification model for predicting hotel booking cancellations. We divided the dataset into training and testing sets, and used the training set to train various classification models, such as Logistic Regression and Random Forest Classifier. We then assessed the performance of these models on the testing set using metrics like ROC.

Our analysis suggests that the dataset is of sufficient size to accurately predict hotel booking cancellations. The selected algorithms demonstrated exceptional performance in achieving the project's objective.

The results of our analysis suggest that classification models can be effective in predicting the cancellation of hotel bookings, providing a valuable tool for both hosts and guests.

Reference:

Alzahrani, M., Alzahrani, A., Alharbi, S., & Alqahtani, F. (2021). Ensemble Learning for Hotel Cancellation Prediction. *International Journal of Intelligent Systems and Applications*, 13(4), 38-47.

Fernandes, N., Vieira, P., & Moro, S. (2020). Predicting Hotel Reservation Cancellations Using Machine Learning. *International Journal of Information Management*, 50, 322-330.

Gao, J., Wang, L., Wang, J., & Zhao, S. (2020). A Deep Learning Approach for Hotel Cancellation Prediction. *Journal of Hospitality and Tourism Technology*, 11(3), 322-333.