FLIGHT PRICE PREDICTION

Submitted by:

BALPREET KAUR

ACKNOWLEDGMENT

This includes mentioning of all the references, research papers, data sources, professionals and other resources that helped you and guided you in completion of the project.

This project was possible thanks to the Fliprobo Team and DataTrained resources.

In addition, here are listed the other resources used:

The references I used are:

- Pandas documentation.
- Libraries documentation like:
- Scikit-learn
- Seaborn
- Used Stack Overflow website for some technical issues.
- Other tutorial websites for some visibility such as:
    - http://indusedu.org/pdfs/IJREISS/IJREISS_3673_21781.pdf
    - https://www.sciencedirect.com/science/article/pii/S131915781830884X
    - https://www.analyticsvidhya.com/blog/2018/10/predicting-stock-price-machine-learningnd-deep-learning-techniques-python/
    - https://medium.com/codex/house-price-prediction-with-machine-learning-in-python-cf9df744f7ff
    - https://www.askpython.com/python/examples/stock-price-prediction-python
    - https://www.alpharithms.com/predicting-stock-prices-with-linear-regression-214618/
    - https://towardsdatascience.com/create-a-model-to-predict-house-prices-using-python-d34fe8fad88f
    - https://data-flair.training/blogs/stock-price-prediction-machine-learning-project-in-python/
    - https://codingshiksha.com/python/python-3-web-scraping-airlines-api-script-to-get-flight-status-using-beautifulsoup4-and-json-module-full-project-for-beginners/
    - https://morioh.com/p/3bf034edaffe
    - https://towardsdatascience.com/how-to-scrape-flight-prices-with-python-using-selenium-a8382a70d5d6
    - https://medium.com/geekculture/flight-price-prediction-using-machine-learning-f18b4bdc70e6
    - https://www.analyticsvidhya.com/blog/2021/06/flight-price-prediction-a-regression-analysis-using-lazy-prediction/
    - https://towardsdatascience.com/feature-engineering-in-python-part-i-the-most-powerful-way-of-dealing-with-data-8e2447e7c69e
    - https://www.aionlinecourse.com/blog/airline-tickets-price-prediction

    - https://towardsdatascience.com/how-to-scrape-flight-prices-with-python-using-selenium-a8382a70d5d6
    - https://ashishmodi27.medium.com/flight-price-prediction-4c5dad31e812
    - https://www.researchgate.net/publication/343434680_Prediction_of_House_Pricing_using_Machine_Learning_with_Python
    - https://colab.research.google.com/github/knightow/mltraining/blob/master/Stock_Price_Prediction_Using_Python_%26_Machine_Learning.ipynb
    - https://blog.quantinsti.com/gold-price-prediction-using-machine-learning-python/
    - https://www.dataquest.io/blog/machine-learning-tutorial/
    - https://www.researchgate.net/publication/335936877_A_Framework_for_Airfare_Price_Prediction_A_Machine_Learning_Approach
    - https://www.analyticsvidhya.com/blog/2021/05/feature-engineering-how-to-detect-and-remove-outliers-with-python-code/#:~:text=%F0%9F%91%89Capping%3A%20In%20this%20technique,dataset%20gives%20that%20capping%20number.

- Books consulted:
    - https://www.researchgate.net/publication/340896273_Machine_Learning_based_Predicting_House_Prices_using_Regression_Techniques
    - https://www.researchgate.net/publication/343241145_Housing_Price_Prediction_via_Improved_Machine_Learning_Techniques
    - https://www.diva-portal.org/smash/get/diva2:1456610/FULLTEXT01.pdf

INTRODUCTION

- Business Problem Framing

The airline industry is considered as one of the most sophisticated industry in using complex pricing strategies. Nowadays,ticket prices can vary dynamically and significantly for the sameflight, even for nearby seats (Etzioni et al., 2003; Narangajavanaet al., 2014). The ticket price of a specific flight can change up to 7 times a day (Etzionietal.,2003) due to demand and companies willing to offer seats and keeping their revenue as high as possible. In addition, competition between airlines makes the task of determining optimal pricing is hard for everyone.

The business problem is that the cheapest available ticket on a given flight gets more and less expensive over time. We have to work on a machine learning model which predict accurately fares of flights.

- Conceptual Background of the Domain Problem

Generally speaking, every data scientist has to have a minimum domain expertise that implies knowledge and understanding of the essential aspects of the specific field of online flight prices and airlines fares available at different moment of time. In other words, a data scientist must know the stuff he/she is involved with. It is essential in order to make and have a valuable results and conclusions. It is essentially required for data mining and pattern discovery. The data scientist needs to know how the essentials of the specific field works so that the discovery, the patterns and evaluation process is guided by an intuitive knowledge of what actually matters both in terms of inputs and outputs as well as of what makes more sense or not.

- Review of Literature

Reviewing the literature, I have got the following information:
There have seen increasing research targeting both customers and airlines point of view in analysing the price of the flights. Customer side researches focus on saving money while airline main focus is to get maximum revenue of the airlines.
Conducted researches employ a variety of techniques ranging from statistical techniques such as regression to different kinds of advanced datamining techniques.
From the customer point of view, determining the minimum price or the best time to buy a ticket is the key issue. The conception of ‘‘tickets bought in advance are cheaper” is no longer working (William Groves and Maria Gini, 2013
The ticket price may be affected by several factors thus may change continuously. To address this, various studies were conducted to support the customer in determining an optimal ticket purchase time and ticket price prediction(AnastasiaLantseva et al.,

2015; Chawla et al., 2017; Domínguez-Mencheroet al., 2014; K. Tziridis et al., 2017; Li et al., 2014; Santana et al.,2017; T. Liu et al., 2017; T.Wohlfarth et al., 2011; V. H et al.,2018; William Groves and Maria Gini, 2013; William Groves andMariaGini, 2015; Y. Chenet al., 2015;Y. XuandJ. Cao, 2017).

As noted by Y. Chen et al. (2015), predicting an optimal ticket purchase time is easier task than predicting the actual ticket price due to various reasons: absence of enough datasets, external factors influencing ticket prices, dynamic behavior of ticket pricing, competition among airlines, proprietary nature of airlines ticket pricing policies etc. Nevertheless, few studies have attempted to predict actual ticket prices with the work done by the authors in (Anastasia Lantseva et al., 2015; Domínguez-Mencheroet al., 2014; K. Tziridis et al., 2017; Santana et al., 2017; T. Liuet al., 2017; V. H et al., 2018) as examples.

On the airlines side, the main goal is to increase revenue and maximize profit. According to Naranga javanaetal., 2014, airlines utilize various kinds of pricing strategies to determine optimal ticket prices: long-term pricing policies, yield pricing which describes the impact of production conditions on ticket prices, and dynamic pricing which is mainly associated with dynamic adjustment of ticket prices in response to various influencing factors. Long term-pricing policies and yield pricing are associated with internal working of the specific airline and do not help that much in predicting dynamic fluctuations in price. On the other hand, dynamic pricing enables a more optimal forecasting of ticket prices based on vibrant factors such as changes in demand and price discrimination (Malighetti et al., 2009).

However, dynamic pricing is challenging as it is highly influenced by various factors including internal factors, external factors, competition among airlines and strategic customers. Internal factors consist of features such as historical ticket price data, ticket purchase date and departure date, season, holidays, supply (number of available airline sand flights), fare class, availability of seats, recent market demand

Therefore, to become profitable in such complex situations, airlines must dynamically adjust ticket prices based on the current demand, the behaviour of customers, ticket prices given by competitors in the market and other internal and external factors (Y. Wang, 2016; Yiwei ChenandVivekF.Farias,2015).

This dynamic adjustment of ticket price response to various influencing actors is known as dynamic

pricing. The aforementioned studies (Malighetti et al., 2009;Narangajavana et al., 2014; Y. Wang, 2016; Yiwei Chen and VivekF. Farias, 2015) explain that dynamic pricing is implemented by airlines as one of the most common price strategies. However, they do not discuss the different kinds of prediction methods that are utilized to implement dynamic pricing. A significant number of research works exit that proposed prediction models for dynamic pricing in airlines which can be classified into two groups: demand prediction (Bo An et al., 2016; Bo Anet al., 2017; Chieh-Hua Wen and Po-

Hung Chen, 2017; DiegoEscobari, 2014; H. Yuan et al., 2014; Jie Liu et al., 2017a,b;M

umbower et al., 2014) and price discrimination (Efthymios Constantinides and Rasha HJ Dierckx, 2014; Mantin Benny andBonwoo Koo, 2010; Marco Alderighi et al., 2011; Steven L.Pullerand Lisa M.Taylor, 2012).

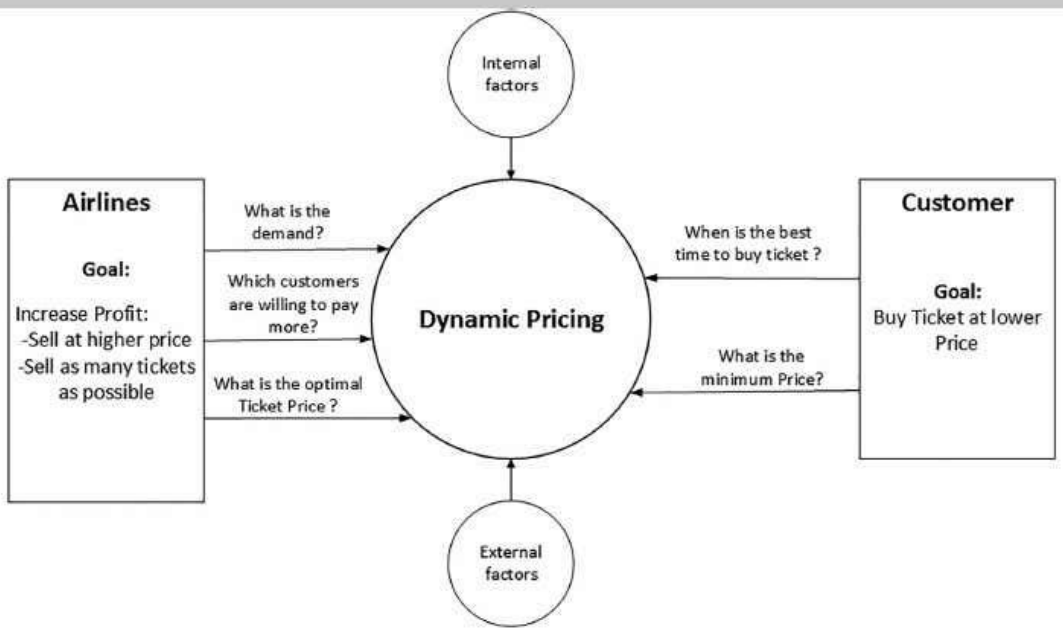


Image from J.A. Abdella et al./Journal of King Saud University – Computer and Information Sciences

- Motivation for the Problem Undertaken

  I was personally interested in Flight Airfare Prediction basically due to help me understand how we can choose the right flight fare and find our optimal flight and find the right flight airfare based on our needs at the time I need to fly.

  The motivation behind it was basically that I understand how to search, scrap, study and predict the best price and choose the best flight option given the conditions.

Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem

```
[18]: #statistical description of the data of features
      df.describe()
```

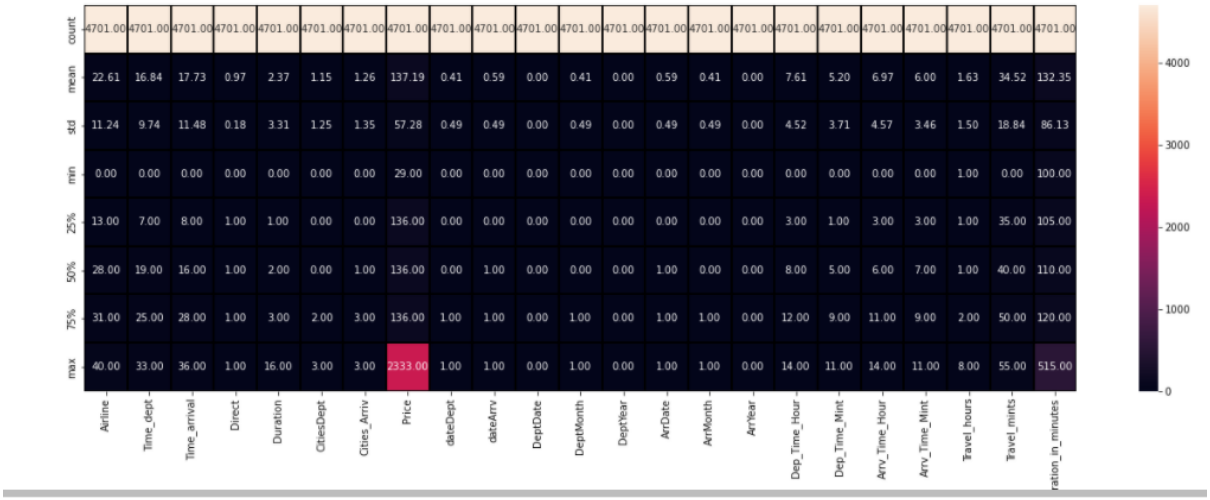
[18]:

	Airline	Time_dept	Time_arrival	Direct	Duration	CitiesDept	Cities_Arriv	Price	dateDept	dateArrv
count	4701	4701	4701	4701	4701	4701	4701	2057	4701	4701
unique	41	34	37	2	17	4	4	215	2	2
top	Vueling	7:00	15:40	directo	1h 50m	BCN Barcelona-El Prat	BCN Barcelona-El Prat	136 €	26/02/2022	05/03/2022
freq	672	540	397	4543	1176	2351	2350	60	2754	2754

Here above we can see the count, unique count, topper and the frecuencies of each feature. If we focus on the target Price we can see it is the only variable that should be integer format. But, since it has € in it, it is described as an object type. that's why all the features description including target Price mentions count, unique, top and freq and does not include info of mean, median,percentiles, etc.

```
[94]: #statistical description of all numerical columns:
      import matplotlib.pyplot as plt
      plt.figure(figsize=(22,7))
      sns.heatmap(df.describe(),annot=True,linewidth=0.1,linecolor="black",fmt="0.2f")
```

[94]: <AxesSubplot:>



Here we can see and highlight the following considerations:

- we have no missing values at this stage as the first row mentions total rows of dataframe we have in all our columns.

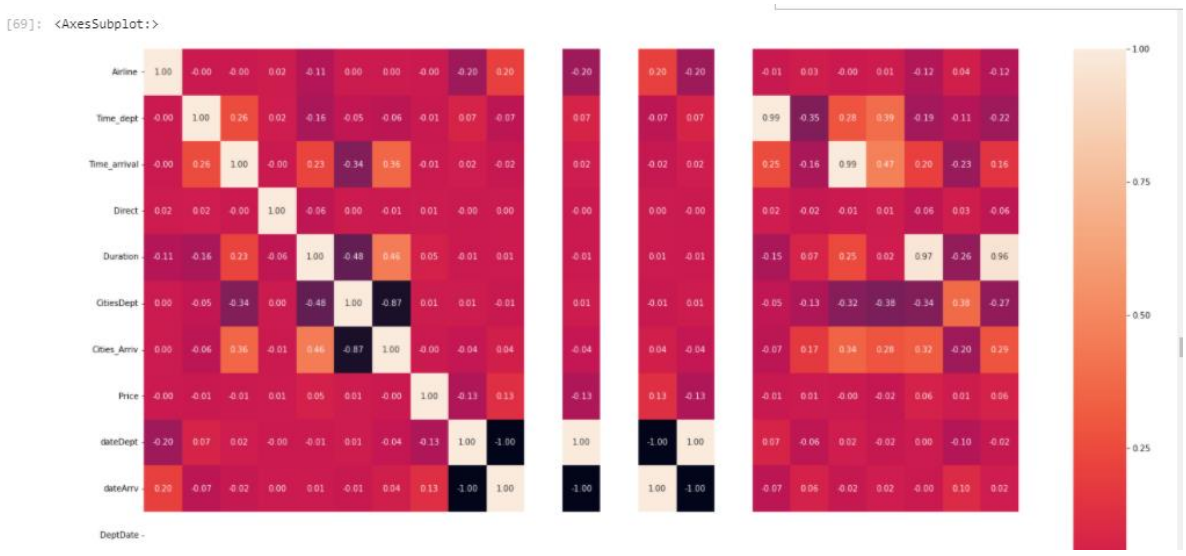
- we have columns like Time_arrival, Duration, CitiesDept, Cities_Arrv,Price, dateDept, DeptMonth, ArrMonth, Dept_Time_Mint, Arrv_Time_Hour, Travel_hours and Duration_in_minutes which have higher mean than the median which hence means that we have right side skewed distribution in these features.

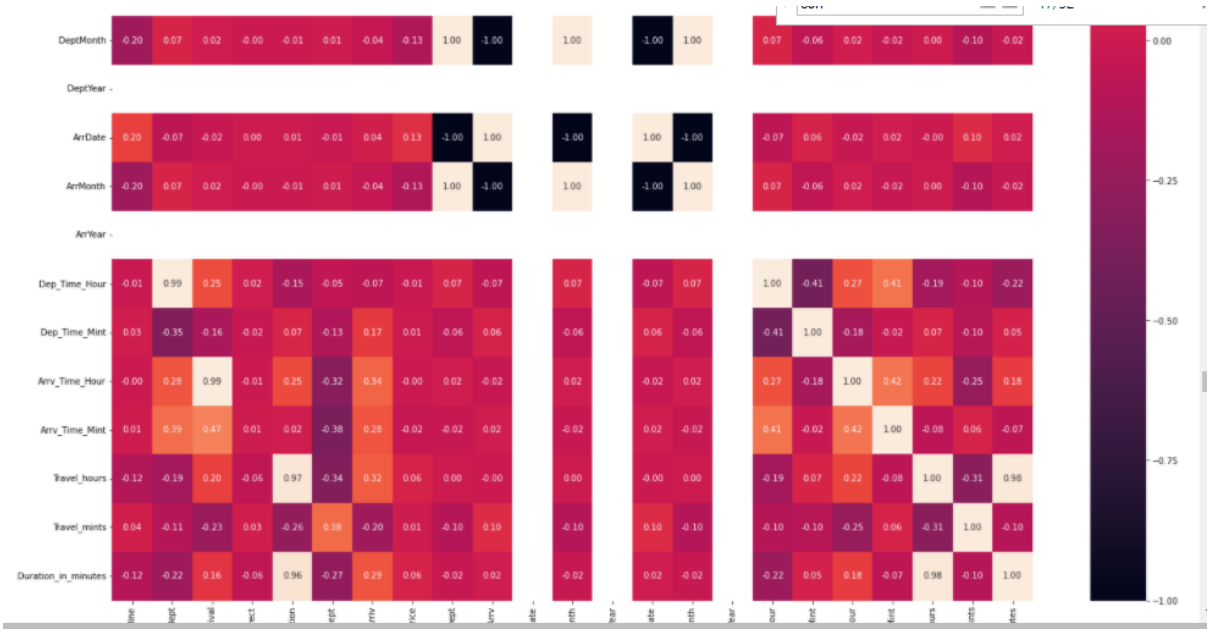
- And DeptDate and DeptYear are non skewed data as there is only 1 unique value. The same happens with Arrival Year.

- The remaining features have left side skewed distribution, which means mean is smaller than the median.

- We have that all columns except Direct, Cities_Ariv, dateDept, dateArriv, Depttdate, DepatMonth, DeptYear, ArrDate, ArrMonth and ArrYear have outliers in them as the max is higher than the 75% percentile.

When checking the correlation between features:





Regarding the target Price, we see the features that are highly correlated with it are the Departure and Arrival Date.

We also see that:

- Departue city and Ariival city are highly correlated with Duration and Arriva Time.

- We also see the departure City and arrival city are also very highly correllated.

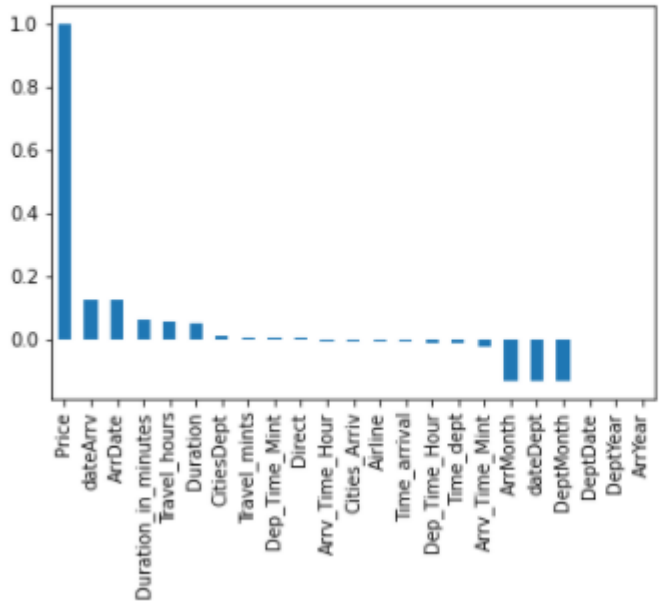
- We see departure time is almost fully correlated with departure time hour. And same happens with Arrival case, we see arrival time is almost fully correlated with Arrival time hour.

- And as it is expected, travel hours is almost fully correlated with duration in minutes and the same time these are almost fully correlated with duration.

Let's now check plot of the correlation of the features with regards the target feature in the following code:

```
[70]: #correlation of independent features with regards the target "Price"
      correlations =df.corr()['Price'].sort_values(ascending=False)
      correlations.plot(kind='bar')
```

```
[70]: <AxesSubplot:>
```



As we can see, departure date and Arrival date have the most impact on our Price target feature. And in third place, we have Duration in minutes, Travel hours and Duration which has considerable impact on our target Price.

After the above analysis, we drop the non-required columns, which we feel have no impact on prices of flights. These columns include all the splits from the date and time of the flights.

Now let's check the features that has most impact on price:

```
[71]: correlations[abs(correlations) > 0.1]
```

```
[71]: Price        1.000000
      dateArrv     0.128731
      ArrDate      0.128731
      ArrMonth    -0.128731
      dateDept    -0.128731
      DeptMonth   -0.128731
      Name: Price, dtype: float64
```

As we said, we have departure date, Arrival date as the main 2 independent features that impacts the most on the Price of the flights.

- Data Sources and their formats

```
17]: #checking and study briefly the dataset sample before we do nothing:
     df.head()
```

17]:

	Airline	Time_dept	Time_arrival	Direct	Duration	CitiesDept	Cities_Arriv	Price	dateDept	dateArrv
0	Vueling, Ryanair	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	29 €	26/02/2022	05/03/2022
1	Vueling, Ryanair	9:00	10:45	directo	1h 45m	BVA Beauvais-Tillé	BCN Barcelona-El Prat	34 €	26/02/2022	05/03/2022
2	Vueling	13:50	15:40	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	NaN	26/02/2022	05/03/2022
3	Vueling	9:00	10:45	directo	1h 45m	BVA Beauvais-Tillé	BCN Barcelona-El Prat	NaN	26/02/2022	05/03/2022
4	Ryanair	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	NaN	26/02/2022	05/03/2022

1. Airline features gives us the flight companies names, which we can use to separate the flight data by different airlines for comparison.

2. The Date of Journey can be used in different ways. As an example, we can say we can separate and analyse the data we have through months or years or even day by day.

3. The CitiesDept and CitiesArrv helps us understand where the flight comes from and went to. :)

4. The Direct feature will inform if the flight is directly landed in the destination airport or not.

5. The Dep time and Arrival Time shows the time of arrival and departure of each flight. So, we can use time to calculate the hours or minutes of Duration or even we can study the flights by time of arrival or departure.

6. The dateDept and dateArrv columns have dates attached along with, which we can separte and these are the cases when the flight takes off from the source on a date and reaches its destination on the same day normally in our case.

7. The Duration is more useful as it calculates the time the flight takes to arrive to their destination.

```
[26]: #Now checking the data.info() so that we can con
      #be useful to check as like data type of each co
      df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4701 entries, 0 to 1946
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Airline       4701 non-null   object
 1   Time_dept     4701 non-null   object
 2   Time_arrival  4701 non-null   object
 3   Direct        4701 non-null   object
 4   Duration      4701 non-null   object
 5   CitiesDept    4701 non-null   object
 6   Cities_Arriv  4701 non-null   object
 7   Price         4701 non-null   int64 
 8   dateDept      4701 non-null   object
 9   dateArrv      4701 non-null   object
dtypes: int64(1), object(9)
memory usage: 404.0+ KB
```

As we can check we have the total number of rows for each feature we have, which means no missing values are left after having applied missing values imputation. Which in the other words means that we have imputed the initial missing values correctly.

Here we can check information of data type and information about number of values present in each column, and data types of each column. We observe that we have all the columns as ‘object’ data types, and only ‘Price’ column (the output) is of integer type.

And we also can check the total of columns and total of rows we have in our dataset. In this we have a total of 4701 entries and total of 11 features including the target variable.

- Data Preprocessing Done

  What were the steps followed for the cleaning of the data? What were the assumptions done and what were the next actions steps over that?

```
[5]: df1 = df1.loc[:, ~df1.columns.str.contains('^Unnamed')] #26/2-5/3 #dropping the unamed column (the first column, the indexed one)
     df1
```

[5]:

	Airline	Time_dept	Time_arrival	Direct	Duration	CitiesDept	Cities_Arriv	Price
0	Vueling, Ryanair	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	29 €
1	Vueling, Ryanair	9:00	10:45	directo	1h 45m	BVA Beauvais-Tillé	BCN Barcelona-El Prat	34 €
2	Vueling	13:50	15:40	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	NaN
3	Vueling	9:00	10:45	directo	1h 45m	BVA Beauvais-Tillé	BCN Barcelona-El Prat	NaN
4	Ryanair	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	NaN
...	...	...	...	...	...	...	...	...
2749	Vueling	12:25	14:05	directo	1h 40m	ORY París-Orly	BCN Barcelona-El Prat	155 €
2750	Transavia France, Vueling	13:50	15:40	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	156 €
2751	Ryanair, Vueling	9:30	11:10	directo	1h 40m	ORY París-Orly	BCN Barcelona-El Prat	329 €
2752	Iberia	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	345 €
2753	Vueling, easyJet	9:30	11:10	directo	1h 40m	ORY París-Orly	BCN Barcelona-El Prat	2333 €

2754 rows × 8 columns

- Dropping the unnamed column (the first column, the indexed one)

```
[6]: df1['dateDept'] = '26/02/2022' #adding new columns od departure date and arrival date
     df1['dateArrv'] = '05/03/2022'
```

```
[7]: df1
```

[7]:

	Airline	Time_dept	Time_arrival	Direct	Duration	CitiesDept	Cities_Arriv	Price	dateDept	dateArrv
0	Vueling, Ryanair	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	29 €	26/02/2022	05/03/2022
1	Vueling, Ryanair	9:00	10:45	directo	1h 45m	BVA Beauvais-Tillé	BCN Barcelona-El Prat	34 €	26/02/2022	05/03/2022
2	Vueling	13:50	15:40	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	NaN	26/02/2022	05/03/2022
3	Vueling	9:00	10:45	directo	1h 45m	BVA Beauvais-Tillé	BCN Barcelona-El Prat	NaN	26/02/2022	05/03/2022
4	Ryanair	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	NaN	26/02/2022	05/03/2022
...	...	...	...	...	...	...	...	...	...	...
2749	Vueling	12:25	14:05	directo	1h 40m	ORY París-Orly	BCN Barcelona-El Prat	155 €	26/02/2022	05/03/2022
2750	Transavia France, Vueling	13:50	15:40	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	156 €	26/02/2022	05/03/2022
2751	Ryanair, Vueling	9:30	11:10	directo	1h 40m	ORY París-Orly	BCN Barcelona-El Prat	329 €	26/02/2022	05/03/2022
2752	Iberia	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	345 €	26/02/2022	05/03/2022
2753	Vueling, easyJet	9:30	11:10	directo	1h 40m	ORY París-Orly	BCN Barcelona-El Prat	2333 €	26/02/2022	05/03/2022

2754 rows × 10 columns

- Adding new columns od departure date and arrival date

Doing the same with the second dataframe:

[11]:
```
df2 #checking the second dataframe
```

[11]:

	Airline	Time_dept	Time_arrival	Direct	Duration	CitiesDept	Cities_Arriv	Price
0	Vueling, Ryanair	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	29 €
1	Vueling	9:00	10:45	directo	1h 45m	BVA Beauvais-Tillé	BCN Barcelona-El Prat	34 €
2	Vueling	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	NaN
3	Vueling	12:25	14:05	directo	1h 40m	ORY París-Orly	BCN Barcelona-El Prat	NaN
4	Ryanair	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	NaN
...	...	...	...	...	...	...	...	...
1942	Air France, Vueling	7:55	9:55	directo	2h 00m	BCN Barcelona-El Prat	CDG Charles de Gaulle	NaN
1943	Air France, easyJet	7:00	8:45	directo	1h 45m	CDG Charles de Gaulle	BCN Barcelona-El Prat	NaN
1944	Vueling, Iberia	9:20	11:30	directo	2h 10m	BCN Barcelona-El Prat	CDG Charles de Gaulle	164 €
1945	Air France, easyJet	12:25	14:05	directo	1h 40m	ORY París-Orly	BCN Barcelona-El Prat	166 €
1946	Air France, Iberia	15:00	17:00	directo	2h 00m	BCN Barcelona-El Prat	CDG Charles de Gaulle	165 €

1947 rows × 8 columns

[12]:
```
df2 = df2.loc[:, ~df2.columns.str.contains('^Unnamed')] #dropping the unamed column (the first column, the indexed one) in the second dataframe
df2
```

[12]:

	Airline	Time_dept	Time_arrival	Direct	Duration	CitiesDept	Cities_Arriv	Price
0	Vueling, Ryanair	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	29 €
1	Vueling	9:00	10:45	directo	1h 45m	BVA Beauvais-Tillé	BCN Barcelona-El Prat	34 €
2	Vueling	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	NaN
3	Vueling	12:25	14:05	directo	1h 40m	ORY París-Orly	BCN Barcelona-El Prat	NaN
4	Ryanair	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	NaN
...	...	...	...	...	...	...	...	...
1942	Air France, Vueling	7:55	9:55	directo	2h 00m	BCN Barcelona-El Prat	CDG Charles de Gaulle	NaN
1943	Air France, easyJet	7:00	8:45	directo	1h 45m	CDG Charles de Gaulle	BCN Barcelona-El Prat	NaN
1944	Vueling, Iberia	9:20	11:30	directo	2h 10m	BCN Barcelona-El Prat	CDG Charles de Gaulle	164 €
1945	Air France, easyJet	12:25	14:05	directo	1h 40m	ORY París-Orly	BCN Barcelona-El Prat	166 €
1946	Air France, Iberia	15:00	17:00	directo	2h 00m	BCN Barcelona-El Prat	CDG Charles de Gaulle	165 €

1947 rows × 8 columns

[13]:
```
df2['dateDept'] = '26/03/2022' #adding new columns od departure date and arrival date in the second dataframe
df2['dateArrv'] = '02/04/2022'
```

[14]:
```
frames=[df1,df2]  #concatenating both dataframes and converting into one dataset
df = pd.concat(frames)
df
```

[14]:

	Airline	Time_dept	Time_arrival	Direct	Duration	CitiesDept	Cities_Arriv	Price	dateDept	dateArrv
0	Vueling, Ryanair	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	29 €	26/02/2022	05/03/2022
1	Vueling, Ryanair	9:00	10:45	directo	1h 45m	BVA Beauvais-Tillé	BCN Barcelona-El Prat	34 €	26/02/2022	05/03/2022
2	Vueling	13:50	15:40	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	NaN	26/02/2022	05/03/2022
3	Vueling	9:00	10:45	directo	1h 45m	BVA Beauvais-Tillé	BCN Barcelona-El Prat	NaN	26/02/2022	05/03/2022
4	Ryanair	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	NaN	26/02/2022	05/03/2022
...	...	...	...	...	...	...	...	...	...	...
1942	Air France, Vueling	7:55	9:55	directo	2h 00m	BCN Barcelona-El Prat	CDG Charles de Gaulle	NaN	26/03/2022	02/04/2022
1943	Air France, easyJet	7:00	8:45	directo	1h 45m	CDG Charles de Gaulle	BCN Barcelona-El Prat	NaN	26/03/2022	02/04/2022
1944	Vueling, Iberia	9:20	11:30	directo	2h 10m	BCN Barcelona-El Prat	CDG Charles de Gaulle	164 €	26/03/2022	02/04/2022
1945	Air France, easyJet	12:25	14:05	directo	1h 40m	ORY París-Orly	BCN Barcelona-El Prat	166 €	26/03/2022	02/04/2022
1946	Air France, Iberia	15:00	17:00	directo	2h 00m	BCN Barcelona-El Prat	CDG Charles de Gaulle	165 €	26/03/2022	02/04/2022

4701 rows × 10 columns

Regarding the departure and arrival date: as I did select both dates automatically through selenium, so adding here the dates manually to our dataframe.

In addition, we also need to mention, we only have flights data for selected departure and arrival dates which are:

- Flights departing on 26 Feb and on 26 March (both Saturday) from Barcelona to Paris (all airports):
    - We have chosen 26th Feb and 26th March just to compare prices of similar flights offers when flying earlier or later.
- Flights departing on 3rd March and 2nd April from Paris (all airports) to Barcelona.
    - For same reason, to compare prices for flights departing soon with prices for flights departing 1 month later.
- We have limited our flights data to 2 cities so that it is easier to compare the prices of 2 cities than cities located in different part of world with different air fares. Also due to the reason that we selected both cities automatically through Selenium automation when web scrapping the data.
- Having all this in consideration, I may say that I decided to go with these 2 cities and dates over 1 month of difference, because this way we can simplify the process of analysing the EDA and also at the same time reach and get our goal which is to understand the price change over a period of time.

Then, we checked for null values:

```
[20]: df.isnull().sum() #checking nulls in our dataset columns
```

```
[20]: Airline            0
      Time_dept          0
      Time_arrival       0
      Direct             0
      Duration           0
      CitiesDept         0
      Cities_Arriv       0
      Price           2644
      dateDept           0
      dateArrv           0
      dtype: int64
```

```
[21]: df[df.isna().any(axis=1)] #checking rows with null values in Price as the Price is the only feature where we have null values as
```

[21]:

	Airline	Time_dept	Time_arrival	Direct	Duration	CitiesDept	Cities_Arriv	Price	dateDept	dateArrv
2	Vueling	13:50	15:40	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	NaN	26/02/2022	05/03/2022
3	Vueling	9:00	10:45	directo	1h 45m	BVA Beauvais-Tillé	BCN Barcelona-El Prat	NaN	26/02/2022	05/03/2022
4	Ryanair	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	NaN	26/02/2022	05/03/2022
5	Vueling, Transavia France	12:25	14:05	directo	1h 40m	ORY París-Orly	BCN Barcelona-El Prat	NaN	26/02/2022	05/03/2022
6	Vueling	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	NaN	26/02/2022	05/03/2022
...	...	...	...	...	...	...	...	...	...	...
1931	Vueling, Iberia	16:40	22:35	directo	5h 55m	BVA Beauvais-Tillé	BCN Barcelona-El Prat	NaN	26/03/2022	02/04/2022
1932	Iberia	7:00	8:50	directo	1h 50m	BCN Barcelona-El Prat	ORY París-Orly	NaN	26/03/2022	02/04/2022
1936	Iberia, Vueling	15:00	17:00	directo	2h 00m	BCN Barcelona-El Prat	CDG Charles de Gaulle	NaN	26/03/2022	02/04/2022

We had 2644 missing values in Price column. Then, we pre-processed and handled the null values we have in our target column Price:

```
[22]: df['Price'] = df['Price'].fillna(df['Price'].mode()[0]) #filling na of target feature Price with mode of the same
      df[df.isna().any(axis=1)]
```

[22]:

Airline	Time_dept	Time_arrival	Direct	Duration	CitiesDept	Cities_Arriv	Price	dateDept	dateArrv

We also changed the data type of target column Price to integer type so that we can analyse our Price values as numerical values:

```
[23]: df['Price'] = (df['Price'].str.strip(' €').astype(int)) #dropping € and converting the target Price feature into a numerical integer type data colum
```

Dropped € and converted the target Price feature into a numerical integer type data column.

```
[24]: df.dtypes #checking to make sure we have Price as integer column.

[24]: Airline         object
      Time_dept       object
      Time_arrival    object
      Direct          object
      Duration        object
      CitiesDept      object
      Cities_Arriv    object
      Price            int64
      dateDept        object
      dateArrv        object
      dtype: object
```

We check in the above screenshot; Price column is changed to integer one.

Also checking we have no more null values:

```
[25]: df.isnull().sum() #checking all null values in Price column are imputed correctly:

[25]: Airline         0
      Time_dept       0
      Time_arrival    0
      Direct          0
      Duration        0
      CitiesDept      0
      Cities_Arriv    0
      Price           0
      dateDept        0
      dateArrv        0
      dtype: int64
```

All null values in Price column are imputed correctly

Then, we did some visualizations and EDA analysis which will be commentared later.

After that, we went to split the departure and arrival dates to extract the ‘Date’, ‘Month’ and ‘Year’ values, and store them in new columns in our dataframe:

```
[24]: df.dateDept=df.dateDept.str.split('/')
      df.dateArrv=df.dateArrv.str.split('/')

[25]: df.dateDept

[25]: 0       [26, 02, 2022]
      1       [26, 02, 2022]
      2       [26, 02, 2022]
      3       [26, 02, 2022]
      4       [26, 02, 2022]
                    ...
      1942    [26, 03, 2022]
      1943    [26, 03, 2022]
      1944    [26, 03, 2022]
      1945    [26, 03, 2022]
      1946    [26, 03, 2022]
      Name: dateDept, Length: 4701, dtype: object

[26]: df.dateArrv

[26]: 0       [05, 03, 2022]
      1       [05, 03, 2022]
      2       [05, 03, 2022]
      3       [05, 03, 2022]
      4       [05, 03, 2022]
                    ...
      1942    [02, 04, 2022]
      1943    [02, 04, 2022]
      1944    [02, 04, 2022]
      1945    [02, 04, 2022]
      1946    [02, 04, 2022]
      Name: dateArrv, Length: 4701, dtype: object

[27]: df['DeptDate']=df.dateDept.str[0]
      df['DeptMonth']=df.dateDept.str[1]
      df['DeptYear']=df.dateDept.str[2]

[28]: df['ArrDate']=df.dateArrv.str[0]
      df['ArrMonth']=df.dateArrv.str[1]
      df['ArrYear']=df.dateArrv.str[2]
```

We did similar separation for Departure time, Arrival Time and Duration: separate hours and minutes.

```
[29]: #we will do the similar separation for Departure time, Arrial Time and Duration: separate hours and minutes.
      df['Time_dept']

[29]: 0        7:00
      1        9:00
      2       13:50
      3        9:00
      4        7:00
              ...
      1942     7:55
      1943     7:00
      1944     9:20
      1945    12:25
      1946    15:00
      Name: Time_dept, Length: 4701, dtype: object
```

[31]:
```
df['Time_dept'] =df.Time_dept.str.split(':')
```

[32]:
```
df['Dep_Time_Hour']=df.Time_dept.str[0]
df['Dep_Time_Mint']=df.Time_dept.str[1]
#df[['Dep_Time_Hour','Dep_Time_Mint']]
```

[34]:
```
df['Time_arrival'] =df.Time_arrival.str.split(':')
```

[35]:
```
df['Arrv_Time_Hour']=df.Time_arrival.str[0]
df['Arrv_Time_Mint']=df.Time_arrival.str[1]
#df[['Dep_Time_Hour','Dep_Time_Mint']]
```

[37]:
```
df.Duration=df.Duration.str.split(' ')
```

[38]:
```
df['Travel_hours']=df.Duration.str[0]
df['Travel_mints']=df.Duration.str[1]
```

[39]:
```
df['Travel_hours']
```

[39]:
```
0       1h
1       1h
2       1h
3       1h
4       1h
       ..
1942    2h
1943    1h
1944    2h
1945    1h
1946    2h
Name: Travel_hours, Length: 4701, dtype: object
```

[40]:
```
df['Travel_hours']=df['Travel_hours'].str.split('h')
df['Travel_hours']=df['Travel_hours'].str[0]
```

[41]:
```
df.Travel_hours=df.Travel_hours
```

[42]:
```
df['Travel_hours']
```

```
[43]: df['Travel_mints']=df.Duration.str[1]
      df['Travel_mints']

[43]: 0       50m
      1       45m
      2       50m
      3       45m
      4       50m
             ...
      1942    00m
      1943    45m
      1944    10m
      1945    40m
      1946    00m
      Name: Travel_mints, Length: 4701, dtype: object

[44]: df['Travel_mints']=df['Travel_mints'].str.split('m')
      df['Travel_mints']=df['Travel_mints'].str[0]

[45]: df['Travel_mints']

[45]: 0       50
      1       45
      2       50
      3       45
      4       50
              ..
      1942    00
      1943    45
      1944    10
      1945    40
      1946    00
      Name: Travel_mints, Length: 4701, dtype: object
```

We could convert travel hours which is duration hours into travel (duration) minutes as follows:

```
[46]: #we will now convert travel hours which is duration hours into travel (duration) minuts:
      df['Travel_hours']=pd.to_numeric(df['Travel_hours'])
      df['Travel_mints']=pd.to_numeric(df['Travel_mints'])
      df['Travel_mints']=df['Travel_mints'].replace(np.NaN,0)
      df['Travel_mints']=df['Travel_mints'].astype('int64')
      df['Duration_in_minutes']=df['Travel_hours']*60+df['Travel_mints']

[47]: df
```

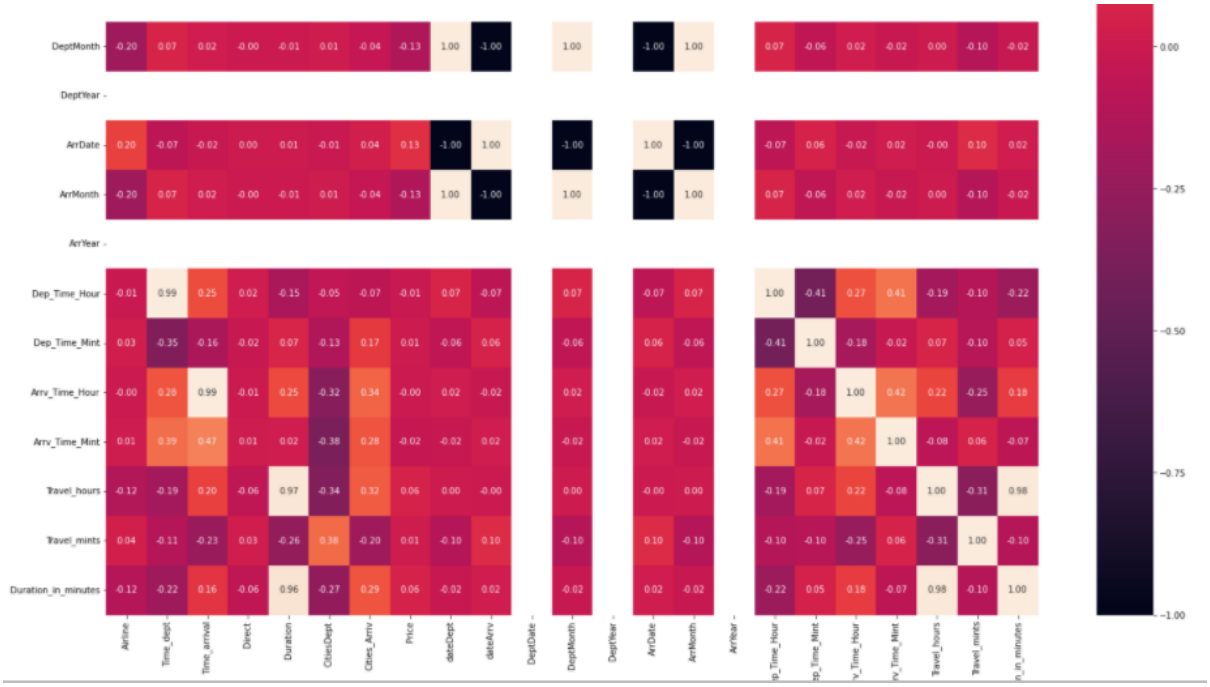
Then having all the columns in place and in the right format, we did label encoding with all the features in order to get correlation calculations and also to dump them in the ML algorithms:

```
[48]: #Label encoding the columns with object type data, which mean all columns except Price which is c

from sklearn.preprocessing import LabelEncoder

for col in df.columns:
        if df[col].dtype == object:
            #df[col] = LabelEncoder().fit_transform(df[col])
            df[col] = LabelEncoder().fit_transform(df[col].astype(str))
```

- Data Inputs- Logic- Output Relationships

  Describe the relationship behind the data input, its format, the logic in between and the output. Describe how the input affects the output.

  The input and output features relationship can be understood easily with through correlation.



From which we could highlight the following highly correlated relations:

Regarding the target Price, we see the features that are highly correlated with it are the Departure and Arrival Date.

We also see that:

- Departure city and Arrival city are highly correlated with Duration and Arriva Time.

- We also see the departure City and arrival city are also very highly correlated.

- We see departure time is almost fully correlated with departure time hour. And same happens with Arrival case, we see arrival time is almost fully correlated with Arrival time hour.

- And as it is expected, travel hours is almost fully correlated with duration in minutes and the same time these are almost fully correlated with duration.

We also got to understand some of the coefficients of the linear regression as shown below:

[257]:
```
#training the model
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train,y_train)
```

[257]:
```
LinearRegression()
```

[258]:
```
#Let's check the Coefficients of linear regression
print('Coefficients: \n', lm.coef_)
```

```
Coefficients:
 [-0.03089345 -0.0277368   3.26498555  3.69694383  3.1729597   0.        ]
```

[259]:
```
#so the coffecients we had for each variable
coeffecients = pd.DataFrame(lm.coef_,X.columns)
coeffecients.columns = ['Coeffecient']
coeffecients
```

[259]:

	Coeffecient
Airline	-0.030893
Time_dept	-0.027737
Direct	3.264986
CitiesDept	3.696944
Cities_Arriv	3.172960
DeptDate	0.000000

➔ Which gave me an accuracy of 0.6876048229225384%

```
# comparing actual values with predicted values
actual_vs_pred = pd.DataFrame({'Actual': y_test, 'Predicted': predictions})
actual_vs_pred
```

[261]:

	Actual	Predicted
0	136	140.192102
1	110	132.520847
2	154	140.130315
3	136	132.135546
4	136	138.361049
...	...	...
1406	144	139.292819
1407	136	135.745982
1408	143	138.453729
1409	136	138.453729
1410	136	136.572056

1411 rows × 2 columns

[262]:
```
#checking the acurracy score
from sklearn.metrics import accuracy_score
print("accuracy_score:",lm.score(X_test, y_test))
```

```
accuracy_score: -0.006876048229225384
```

- State the set of assumptions (if any) related to the problem under consideration

  Here, you can describe any presumptions taken by you.

  Regarding the departure departure and arrival date: as I did select both dates automatically through selenium, so adding here the dates manually to our dataframe.

  In addition, we also need to mention, we only have flights data for selected departure and arrival dates which are:

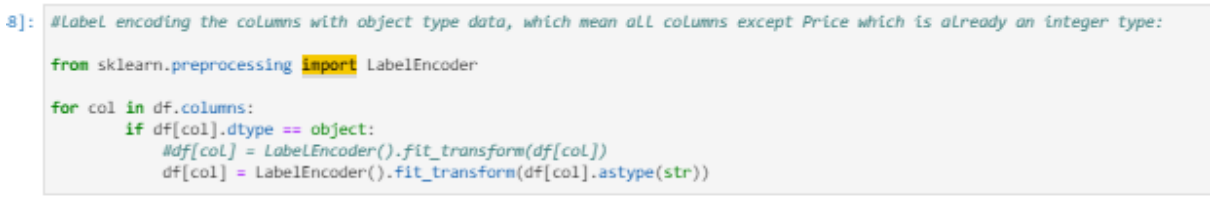
- Flights departing on 26 Feb and on 26 March (both Saturday) from Barcelona to Paris (all airports):
  - We have chosen 26th Feb and 26th March just to compare prices of similar flights offers when flying earlier or later.

- Flights departing on 3rd March and 2nd April from Paris (all airports) to Barcelona.
    - For same reason, to compare prices for flights departing soon with prices for flights departing 1 month later.
- We have limited our flights data to 2 cities so that it is easier to compare the prices of 2 cities than cities located in different part of world with different air fares. Also due to the reason that we selected both cities automatically through Selenium automation when web scrapping the data.
- Having all this in consideration, I may say that I decided to go with these 2 cities and dates over 1 month of difference, because this way we can simplify the process of analysing the EDA and also at the same time reach and get our goal which is to understand the price change over a period of time.

• Hardware and Software Requirements and Tools Used

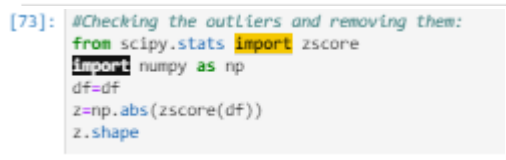
Listing down the hardware and software requirements along with the tools, libraries and packages used. Describe all the software tools used along with a detailed description of tasks done with those tools.

```
[2]: # importing required libraries.
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     %matplotlib inline
```



```
8]: #Label encoding the columns with object type data, which mean all columns except Price which is already an integer type:

    from sklearn.preprocessing import LabelEncoder

    for col in df.columns:
            if df[col].dtype == object:
                #df[col] = LabelEncoder().fit_transform(df[col])
                df[col] = LabelEncoder().fit_transform(df[col].astype(str))
```

```
i2]: #Let's check skewness
     from scipy.stats import skew
     df=df
     skew_list = skew(df[nf],nan_policy='omit') #sending all numericalfeatures and omitting nan values
     skew_list_df = pd.concat([pd.DataFrame(nf,columns=['Features']),pd.DataFrame(skew_list,columns=['Skewness'])],axis = 1)
```



```
[73]: #Checking the outliers and removing them:
      from scipy.stats import zscore
      import numpy as np
      df=df
      z=np.abs(zscore(df))
      z.shape
```

```
[80]: #Transforming the data to remove the skewness of our features excluding target feature:
      from sklearn.preprocessing import power_transform
      x=power_transform(X,method='yeo-johnson')
```

```
]: #now we will standardrize the data of our features excluding target feature:
   from sklearn.preprocessing import StandardScaler
   sc=StandardScaler()
   x=sc.fit_transform(x)
   x
```

```
: # Import library for VIF to check the multicolinearity:
  from statsmodels.stats.outliers_influence import variance_inflation_factor
  def CalculateVIF(Data):
   # Calculating VIF
       vif = dict()
       vif["FeatureColumns"] = Data.columns
       vif["VIF"] = [variance_inflation_factor(Data.values, i) for i in range(Data.shape[1])]
       return(pd.DataFrame(vif))


  CalculateVIF(df_new).sort_values(by='VIF',ascending=False)
```

Import library for VIF to check the multicollinearity.

```
: #Modeling-- OKRR
  #We now proceed to the main step of our machine learning, fitting the model and predicting

  #Importing libraries

  from sklearn.linear_model import LinearRegression
  from sklearn.tree import DecisionTreeRegressor
  from sklearn.neighbors import KNeighborsRegressor
  from sklearn.svm import SVR
  from sklearn.model_selection import train_test_split
  from sklearn.metrics import r2_score, mean_absolute_error,mean_squared_error
```

```
3]: #cheching the differents alrorithms and compare their results:

   from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
   from sklearn.ensemble import AdaBoostRegressor

   dt=DecisionTreeRegressor()
```

```
: #Regularization: importing libraries:
  from sklearn.model_selection import GridSearchCV
  from sklearn.model_selection import cross_val_score
  import warnings
  warnings.filterwarnings('ignore')
```

```
  from sklearn.model_selection import RepeatedKFold
```

```
: #Last try doing Cross Validation with model GBR, eventhough we know we wont have bet
  from sklearn.model_selection import RandomizedSearchCV
```

```
: #we will cross validate with Elastic Net algorithm:
  from sklearn.linear_model import ElasticNet
```

```
: #we will cross validate with Lasso techniquue
  from sklearn.linear_model import Lasso
```

I used Jupyter-notebook and Python 3 language.

Imported required libraries such as:

- pandas
- numpy
- matplotlib.pyplot
- seaborn
- sklearn
-statsmodels

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more.

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Sklearn:

- ➔ Simple and efficient tools for predictive data analysis
- ➔ Accessible to everybody, and reusable in various contexts
- ➔ Built on NumPy, SciPy, and matplotlib
- ➔ Open source, commercially usable - BSD license

statsmodels is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration. An extensive list of result statistics are available for each estimator. The results are tested against existing statistical packages to ensure that they are correct.

For modelling, we imported different algorithms, pre-processing, splitting tools, data transformation and metrics from different libraries:

- From sklearn.linear_model imported LinearRegression
- From sklearn.linear_model import Lasso and ElasticNet (Lasso is a type of regression that uses shrinkage and ElasticNet is Linear regression with combined L1 and L2 priors as regularizer.)
- From sklearn.tree imported DecisionTreeRegressor
- From sklearn.neighbors imported KNeighborsRegressor
- From sklearn.neighbors imported LocalOutlierFactor (Unsupervised Outlier Detection using the Local Outlier Factor (LOF).

- From sklearn.svm imported SVR
- From sklearn.svm imported OneClassSVM (for Unsupervised Outlier Detection)
- From sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor
- From sklearn.ensemble import IsolationForest (Return the anomaly score of each sample using the IsolationForest algorithm)

All these previous imports were executed in order to fit and train the data.

- From sklearn.model_selection imported train_test_split

The train_test_split was used to split the data in the training and testing set.

- From sklearn.preprocessing import LabelEncoder
  LabelEncoder helped me to convert the object type data to numerical for statistic analysis including correlations, plotting purposes and machine learning algorithm training.
- From scipy.stats import skew → for skewness calculation.
- From statsmodels.stats.outliers_influence import variance_inflation_factor → to calculate VIF and check multicollinearity.
- From scipy.stats import zscore → to calculate zscore and check the outliers.
- From sklearn.preprocessing import power_transform→ to transform and reduce the skewness.
- From sklearn.preprocessing import StandardScaler→ to normalize the data and put it in the same scale for better comparison.
- From sklearn.metrics import r2_score, mean_absolute_error,mean_squared_error→ metrics error to calculate for algorithm comparison and will help us to select the best algorithm for our case.
- From sklearn.model_selection import GridSearchCV
- From sklearn.model_selection import RandomizedSearchCV

The gridsearch CV and randomizedsearch CV is used to get the best CV we can get in order to get the best scores through our best algorithm.

- From sklearn.model_selection import cross_val_score

Cross_val_score helps us to calculate the score of the cross validation.

- Executed %matplotlib inline to call the predefined functions that will plot and default style the plot for you.

Also did import warnings in order to ignore them:

import warnings

warnings.filterwarnings('ignore')

And also imported pickle for the model saving.

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)

  The main approach I decided to follow is to use linear algorithm
  and a few other regressions algorithms as the model should do a prediction of a continuous target variable.

- Testing of Identified Approaches (Algorithms)

- LinearRegression
- Lasso and ElasticNet (Lasso is a type of regression that uses shrinkage and ElasticNet is Linear regression with combined L1 and L2 priors as regularizer.)
- DecisionTreeRegressor
- KNeighborsRegressor
- SVR
- RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor
- For outliers handling, we did use ensemble like IsolationForest, from sklearn.neighbor we used LocalOutlierFactor and from SVM, OneClassSVM.

- Run and Evaluate selected models

  Describe all the algorithms used along with the snapshot of their code and what were the results observed over different evaluation metrics.

Then, after checking and doing some statistical descriptive analysis, we decided starting with simple Linear regression modelling.

[96]:
```
#Modeling-- OKRR
#We now proceed to the main step of our machine learning, fitting the model and predicting the output

#Importing Libraries

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error,mean_squared_error
```

[97]:
```
# evaluate model performance with outliers removed using isolation forest
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import IsolationForest
from sklearn.metrics import mean_absolute_error
'''# load the dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
df = read_csv(url, header=None)
# retrieve the array
data = df.values
# split into input and output elements
X, y = data[:, :-1], data[:, -1]'''
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(x, Y, test_size=0.33, random_state=1)
# summarize the shape of the training dataset
print(X_train.shape, y_train.shape)
# identify outliers in the training dataset
iso = IsolationForest(contamination=0.1)
yhat = iso.fit_predict(X_train)
# select all rows that are not outliers
mask = yhat != -1
#X_train, y_train = X_train[mask, :], y_train[mask]
# summarize the shape of the updated training dataset
print(X_train.shape, y_train.shape)
# fit the model
model = LinearRegression()
model.fit(X_train, y_train)
# evaluate the model
yhat = model.predict(X_test)
# evaluate predictions
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)
```

```
(3149, 13) (3149,)
(3149, 13) (3149,)
MAE: 22.302
```

LinearRegression fits a linear model with coefficients w = (w1, ..., wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation. And for results comparison, we calculated R2 score, MAE and RMSE as follows:

[101]:
```
x_train=X_train
y_train=y_train
x_test=X_test
model.fit(x_train,y_train)
pred=model.predict(x_test)
test_score=r2_score(y_test,pred)
train_score=r2_score(y_train,model.predict(x_train))
if abs(train_score-test_score)<=0.3:
    print(model)
    print(abs(train_score-test_score))
    print("R2 score is: ", r2_score(y_test,pred))
    print("R2 score for train data: ", r2_score(y_train,model.predict(x_train)))
    print("Mean absolute error is: ", mean_absolute_error(y_test, pred))
    print("Mean squared error is: ", mean_squared_error(y_test, pred))
    print("Root mean squared error is: ", (mean_squared_error(y_test,pred)))
```

```
LinearRegression()
0.005578364089641075
R2 score is:  0.02351419916815012
R2 score for train data:  0.017935835078509044
Mean absolute error is:  22.302395673754685
Mean squared error is:  3000.4058698973145
Root mean squared error is:  3000.4058698973145
```

We tried to find best Cross validation and checked the r2_score for comparison as follows:

[95]:
```
#Building Machine Learning Models
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
for i in range(0,10):
    X_train, X_test, Y_train, Y_test = train_test_split(x, Y, test_size=0.25, random_state=i)
    lr.fit(X_train,Y_train)
    pred_train=lr.predict(X_train)
    pred_test=lr.predict(X_test)
    print(f"At random state {i}, the training accuracy is:  {r2_score(Y_train,pred_train)}")
    print(f"At random state {i}, the testing accuracy is:  {r2_score(Y_test,pred_test)}")
    print("\n")
```

```
At random state 0, the training accuracy is:  0.0317715113797401
At random state 0, the testing accuracy is:  0.008024993151128923


At random state 1, the training accuracy is:  0.018824193194333705
At random state 1, the testing accuracy is:  0.029856157207222456


At random state 2, the training accuracy is:  0.02318492929843785
At random state 2, the testing accuracy is:  0.005123130669723142


At random state 3, the training accuracy is:  0.031570207897615776
At random state 3, the testing accuracy is:  0.007989100466764931


At random state 4, the training accuracy is:  0.021182252826186798
At random state 4, the testing accuracy is:  0.026535275093585442


At random state 5, the training accuracy is:  0.018657738173366134
At random state 5, the testing accuracy is:  0.03127120643709047


At random state 6, the training accuracy is:  0.02347001169473495
At random state 6, the testing accuracy is:  0.011746706233870419


At random state 7, the training accuracy is:  0.02360270473452597
At random state 7, the testing accuracy is:  0.006081836971225774


At random state 8, the training accuracy is:  0.022040225830584292
At random state 8, the testing accuracy is:  0.021326775764635153
```

[102]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
#X_train, X_test, Y_train, Y_test = train_test_split(x, Y, test_size=0.25, random_state=5)
lr.fit(X_train,y_train)
```

[102]:
```
LinearRegression()
```

[97]:
```
#checking the R2 Score for the linear regression:
from sklearn.metrics import r2_score
pred_test=lr.predict(X_test)
print(r2_score(Y_test,pred_test))
```

```
0.03127120643709047
```

[104]:
```
#checking the R2 Score for the linear regression:
from sklearn.metrics import r2_score
pred_test=lr.predict(X_test)
print(r2_score(y_test,pred_test))
```

```
0.02351419916815012
```

[ ]:
```
# evaluate the model thorugh mean absolute error
yhatlr = lr.predict(X_test)
# evaluate predictions
maelr = mean_absolute_error(y_test[0:1176], yhatlr)
print('MAE: %.3f' % maelr)
```

```
MAE: 23.508
```

[107]:
```
# evaluate the model thorugh mean absolute error
yhatlr = lr.predict(X_test)
# evaluate predictions
maelr = mean_absolute_error(y_test, yhatlr)
print('MAE: %.3f' % maelr)
```

```
MAE: 22.302
```

We also did calculate R2_Score, MAE, RMSE for the same:

[108]:
```
print(lr)
print("R2 score is: ", r2_score(y_test,yhatlr))
print("R2 score for train data: ", r2_score(y_train,lr.predict(x_train)))
print("Mean absolute error is: ", mean_absolute_error(y_test, yhatlr))
print("Mean squared error is: ", mean_squared_error(y_test, yhatlr))
print("Root mean squared error is: ", (mean_squared_error(y_test,yhatlr)))
```

```
LinearRegression()
R2 score is:  0.02351419916815012
R2 score for train data:  0.017935835078509044
Mean absolute error is:  22.302395673754685
Mean squared error is:  3000.4058698973145
Root mean squared error is:  3000.4058698973145
```

[115]:
```
#creating model instance with best params for GBR model:
model=GradientBoostingRegressor(alpha=0.098,
                                learning_rate=0.1,
                                max_depth=2,
                                min_samples_split=2,
                                min_samples_leaf=1,
                                n_estimators=100)

model.fit(x_train,y_train)
pred=model.predict(x_test)
test_score=r2_score(y_test,pred)
train_score=r2_score(y_train,model.predict(x_train))
if abs(train_score-test_score)<=0.3:
    print(rfr2)----------------------------------------------------------------------------->delete the outputs of this print, its GBR model not RF
    print(abs(train_score-test_score))
    print("R2 score is: ", r2_score(y_test,pred))
    print("R2 score for train data: ", r2_score(y_train,model.predict(x_train)))
    print("Mean absolute error is: ", mean_absolute_error(y_test, pred))
    print("Mean squared error is: ", mean_squared_error(y_test, pred))
    print("Root mean squared error is: ", (mean_squared_error(y_test,pred)))
```

```
RandomForestRegressor(n_estimators=10)
0.011449948311814562
R2 score is:  0.9537134890052974
R2 score for train data:  0.965163437317112
Mean absolute error is:  400.82168588558903
Mean squared error is:  364747.4337817024
Root mean squared error is:  364747.4337817024
```

The r2_score received for Gradient Boosting Regressor comes out to be better after hypertuning, which is 95,37%, as compared to Random Forest Regressor giving accuracy as 94,46%. The value of MAE also decreases, signifying that we were able to tune our model as we can see below. Hence we select Gradient Boosting Regressor as our final model, save the model using best parameters, and create model object using pickle.

Then we tried using LASSO for regularization:

Lasso is the same Linear Model trained with L1 prior as regularizer (aka the Lasso).

The optimization objective for Lasso is:

(1 / (2 * n_samples)) * ||y - Xw||^2_2 + alpha * ||w||_1

Technically the Lasso model is optimizing the same objective function as the Elastic Net with l1_ratio=1.0 (no L2 penalty).

Lasso regularization penalizes the sum of the absolute value of the regression coefficients ($||\beta||1=\sum pj=1|\beta j|$) . This penalty is known as l1 and has the effect of forcing the coefficients of the predictors to approach zero. Since a predictor with zero regression coefficient does not influence the model, lasso manages to exclude less relevant predictors. As in ridge, the degree of penalty is controlled by the hyperparameter λ . When $\lambda=0$, the result is equivalent to that of a linear model by ordinary least squares. As λ increases, the penalty is greater and more predictors are excluded.

[167]:
```
#Regularization: importing libraries:
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings('ignore')
```

[84]:
```
#we will cross validate with Lasso techniquue
from sklearn.linear_model import Lasso
parameters={ 'alpha':[.0001,.001,.01,.1,1,10], 'random_state': list(range(0,10))}
ls=Lasso()
clf=GridSearchCV(ls,parameters)
clf.fit(X_train,Y_train)
print(clf.best_params_)
```

```
{'alpha': 1, 'random_state': 0}
```

[116]:
```
#we will cross validate with Lasso techniquue
from sklearn.linear_model import Lasso
parameters={ 'alpha':[.0001,.001,.01,.1,1,10], 'random_state': list(range(0,10))}
ls=Lasso()
clf=GridSearchCV(ls,parameters)
clf.fit(X_train,y_train)
print(clf.best_params_)
```

```
{'alpha': 1, 'random_state': 0}
```

[85]:
```
#checking the r2 score with Lasso technique:
ls=Lasso(alpha=1,random_state=0)
ls.fit(X_train,Y_train)
ls.score(X_train,Y_train)
pred_ls=ls.predict(X_test)
lss=r2_score(Y_test,pred_ls)
lss
```

[85]:
```
0.024428237647289067
```

[119]:
```
#checking the r2 score with Lasso technique:
ls=Lasso(alpha=1,random_state=0)
ls.fit(X_train,y_train)
ls.score(X_train,y_train)
pred_ls=ls.predict(X_test)
lss=r2_score(y_test,pred_ls)
lss
```

[119]:
```
0.017532126090745503
```

```
[86]: #Let's try with cross fold 15, for example, as the accuracies dont change that much with CV number
      cv_score=cross_val_score(ls,x,Y,cv=15)
      cv_mean=cv_score.mean()
      cv_mean

[86]: -0.7257952177143091

[120]: #Let's try with cross fold 15, for example, as the accuracies dont change that much with CV number
       cv_score=cross_val_score(ls,x,Y,cv=15)
       cv_mean=cv_score.mean()
       cv_mean

[120]: -0.6655685454738893

[121]: #Let's try with cross fold 15, for example, as the accuracies dont change that much with CV number
       cv_score=cross_val_score(lr,x,Y,cv=15)
       cv_mean=cv_score.mean()
       cv_mean

[121]: -0.8688581170731605

[88]: # evaluate the model
      yhatls = ls.predict(X_test)
      # evaluate predictions
      maels = mean_absolute_error(y_test[0:1176], yhatls)
      print('MAE: %.3f' % maels)

      MAE: 22.273

[123]: # evaluate the model
       yhatls = ls.predict(X_test)
       # evaluate predictions
       maels = mean_absolute_error(y_test, yhatls)
       print('MAE: %.3f' % maels)

       MAE: 21.219
```

And compared it also through the same metrics:

```
[124]: print(ls)
       print("R2 score is: ", r2_score(y_test,yhatls))
       print("R2 score for train data: ", r2_score(y_train,ls.predict(x_train)))
       print("Mean absolute error is: ", mean_absolute_error(y_test, yhatls))
       print("Mean squared error is: ", mean_squared_error(y_test, yhatls))
       print("Root mean squared error is: ", (mean_squared_error(y_test,yhatls)))

       Lasso(alpha=1, random_state=0)
       R2 score is:  0.017532126090745503
       R2 score for train data:  0.015509033892896928
       Mean absolute error is:  21.219165535592868
       Mean squared error is:  3018.78672823679
       Root mean squared error is:  3018.78672823679

[89]: #Let's try with Ensemble Technique including Random ForestRegressor:
```

Then we tried with Ensemble Technique RandomForestRegressor:

A random forest regressor.
A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.
Prediction based on the trees is more accurate because it takes into account many predictions. This is because of the average value used. These algorithms are more stable because any changes in dataset can impact one tree but not the forest of trees.

[89]:
```
#Let's try with Ensemble Technique including Random ForestRegressor:
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

parameters={'criterion':['mse','mae'],'max_features':["auto","sqrt","log2"]}
rf=RandomForestRegressor()
clf=GridSearchCV(rf,parameters)
clf.fit(X_train,Y_train)
print(clf.best_params_)
```

```
{'criterion': 'mae', 'max_features': 'sqrt'}
```

[90]:
```
#Checking the R2 Score on CV of Rfr:
rf=RandomForestRegressor(criterion="mae",max_features="sqrt")
rf.fit(X_train,Y_train)
rf.score(X_train,Y_train)
pred_decision=rf.predict(X_test)

rfs=r2_score(Y_test,pred_decision)
print('R2 score:',rfs*100)

rfscore=cross_val_score(rf,x,Y, cv=5)
rfc=rfscore.mean()
print('Cross Val Score:', rfc*100)

pred_test=rf.predict(X_test)
print(r2_score(Y_test,pred_test))
```

```
R2 score: -9.78978242952957
Cross Val Score: -163.61147925328842
-0.0978978242952957
```

And got the previous score and the following MAE:

[91]:
```
# evaluate the model
yhatrf = rf.predict(X_test)
# evaluate predictions
maerf = mean_absolute_error(y_test[0:1176], yhatrf)
print('MAE: %.3f' % maerf)
```

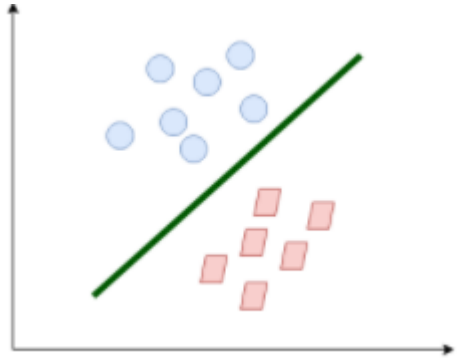
```
MAE: 26.899
```

Then I also tried to compare few ML algorithms which were:

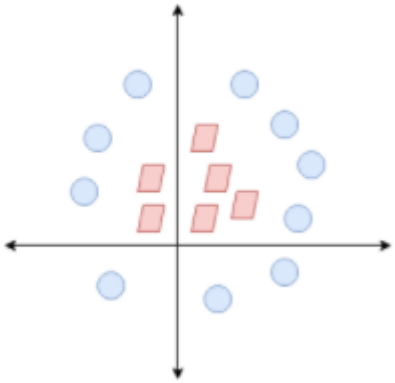
- SVR
- KNN
- LR
- RFR
- ADAB
- GBR

So what exactly is Support Vector Machine (SVM)? We'll start by understanding SVM in simple terms. Let's say we have a plot of two label classes as shown in the figure below:

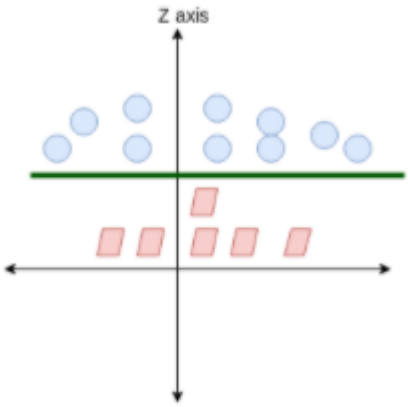
Can you decide what the separating line will be? You might have come up with this:



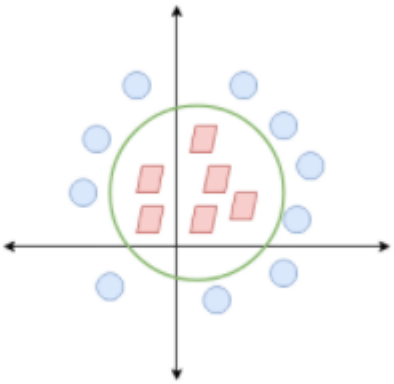
The line fairly separates the classes. This is what SVM essentially does – **simple class separation.** Now, what is the data was like this:



Here, we don't have a simple line separating these two classes. So we'll extend our dimension and introduce a new dimension along the z-axis. We can now separate these two classes:



When we transform this line back to the original plane, it maps to the circular boundary as I've shown here:



This is exactly what SVM does! It tries to find a line/hyperplane (in multidimensional space) that separates these two classes. Then it classifies the new point depending on whether it lies on the positive or negative side of the hyperplane depending on the classes to predict.

KNN:

> In statistics, the k-nearest neighbors algorithm (k-NN) is a non-parametric classification method first developed by Evelyn Fix and Joseph Hodges in 1951,[1] and later expanded by Thomas Cover.[2] It is used for classification and regression. In both cases, the input consists of the k closest training examples in a data set. The output depends on whether k-NN is used for classification or regression:

In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

ADAB:

An AdaBoost [1] regressor is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction. As such, subsequent regressors focus more on difficult cases.

GBR:

Gradient boosting is a machine learning technique used in regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees.[1][2] When a decision tree is the weak learner, the resulting algorithm is called gradient-boosted trees; it usually outperforms random forest.[1][2][3] A gradient-boosted trees model is built in a stage-wise fashion as in other boosting methods, but it generalizes the other methods by allowing optimization of an arbitrary differentiable loss function.

Gradient Boosting for regression.

GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

And here you have snapshot of the results comparison:

[93]:
```
#cheching the differents alrorithms and compare their results:

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.ensemble import AdaBoostRegressor

dt=DecisionTreeRegressor()
svr=SVR()
knn=KNeighborsRegressor()
lr=LinearRegression()
rfr=RandomForestRegressor()
adaB=AdaBoostRegressor()
gbr=GradientBoostingRegressor()

x_train, x_test, y_train, y_test=train_test_split(x,Y,test_size=0.25,random_state=42)
for i in [dt,svr,knn,lr,rfr,adaB,gbr]:
    i.fit(x_train,y_train)
    pred=i.predict(x_test)
    test_score=r2_score(y_test,pred)
    train_score=r2_score(y_train,i.predict(x_train))
    if abs(train_score-test_score)<=0.1:
        print(i)
        print("R2 score is: ", r2_score(y_test,pred))
        print("R2 score for train data: ", r2_score(y_train,i.predict(x_train)))
        print("Mean absolute error is: ", mean_absolute_error(y_test, pred))
        print("Mean squared error is: ", mean_squared_error(y_test, pred))
        print("Root mean squared error is: ", (mean_squared_error(y_test,pred)))
```

```
SVR()
R2 score is:  -0.00048495738410014866
R2 score for train data:  -9.270002734029781e-05
Mean absolute error is:  17.664367116532134
Mean squared error is:  1980.9366347285354
Root mean squared error is:  1980.9366347285354
LinearRegression()
R2 score is:  0.020945038140180894
R2 score for train data:  0.02203941808467058
Mean absolute error is:  21.8607403032321
Mean squared error is:  1938.5057486839198
Root mean squared error is:  1938.5057486839198
```

Then, we tried several times these different algorithms but got no better results.

[103]:
```
#Last try with GBR model:
model=GradientBoostingRegressor(alpha=0.098,
                                learning_rate=0.1,
                                max_depth=5,
                                min_samples_split=2,
                                min_samples_leaf=1,
                                n_estimators=100)

model.fit(x_train,y_train)
pred=model.predict(x_test)
test_score=r2_score(y_test,pred)
train_score=r2_score(y_train,model.predict(x_train))
if abs(train_score-test_score)<=0.3:
    print(model)
    print(abs(train_score-test_score))
    print("R2 score is: ", r2_score(y_test,pred))
    print("R2 score for train data: ", r2_score(y_train,model.predict(x_train)))
    print("Mean absolute error is: ", mean_absolute_error(y_test, pred))
    print("Mean squared error is: ", mean_squared_error(y_test, pred))
    print("Root mean squared error is: ", (mean_squared_error(y_test,pred)))
```

[104]:
```
# evaluate the model
yhatmodel = model.predict(X_test)
# evaluate predictions
maemodel = mean_absolute_error(y_test, yhatmodel)
print('MAE: %.3f' % maemodel)
```

```
MAE: 24.103
```

```
gbr=GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                          learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None,
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)

gbr.fit(x_train,y_train)
pred=gbr.predict(x_test)
test_score=r2_score(y_test,pred)
train_score=r2_score(y_train,gbr.predict(x_train))
if abs(train_score-test_score)<=0.1:
    print(gbr)
    print("R2 score is: ", r2_score(y_test,pred))
    print("R2 score for train data: ", r2_score(y_train,gbr.predict(x_train)))
    print("Mean absolute error is: ", mean_absolute_error(y_test, pred))
    print("Mean squared error is: ", mean_squared_error(y_test, pred))
    print("Root mean squared error is: ", (mean_squared_error(y_test,pred)))
```

Random Forest model gives us the best accuracy, with an R2 score of 88.7%, but the model is overfitting
Gradient boosting also gives a score of 82%, which is better than K-Neighbors and the model is not ove

Cross Validation We will perform the cross validation of our 2 model to check if the models have any ov

[157]:
```
#checking the best cross validation for GBR and checking the mean of CV:
from sklearn.model_selection import cross_val_score
for i in range(2,9):
    cv=cross_val_score(gbr,x,Y,cv=i)
    print(gbr,cv.mean()) #got no better results
```

```
GradientBoostingRegressor() -9.897327581229675
GradientBoostingRegressor() -1.4731915101668325
GradientBoostingRegressor() -1.0172863281500484
GradientBoostingRegressor() -0.9604483928158152
GradientBoostingRegressor() -0.9875019444561461
GradientBoostingRegressor() -0.9489866255274358
GradientBoostingRegressor() -0.945125705020511 5
```

[158]:
```
#checking the best cross validation for RFR and checking the mean of CV:
from sklearn.model_selection import cross_val_score
for i in range(2,9):
    cv=cross_val_score(rfr,x,Y,cv=i)
    print(rfr,cv.mean()) #got no better results
```

```
RandomForestRegressor() -14.056728728316902
RandomForestRegressor() -2.076436937872806
RandomForestRegressor() -1.993533190792761
RandomForestRegressor() -1.61854753390637
RandomForestRegressor() -1.6911172321439267
RandomForestRegressor() -1.475285225103622
RandomForestRegressor() -1.671315014397226
```

[105]:
```
#Last try doing Cross Validation with model GBR, eventhough we know we wont have better Results than Linear Regression
from sklearn.model_selection import RandomizedSearchCV
n_estimators=[int(x) for x in np.linspace(start=100, stop=1200, num=6)]
max_depth=[int(x) for x in np.linspace(start=5, stop=30, num=4)]

random_grid={
    'n_estimators': n_estimators,
    'max_features': ['auto','sqrt'],
    'max_depth':max_depth,
    'min_samples_split':[5,10,15,100]}

rf_random=RandomizedSearchCV(estimator= model, param_distributions= random_grid, cv=3, verbose=2, n_jobs=-1)
rf_random.fit(X_train, y_train)
rf_random.best_params_
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

[105]:
```
{'n_estimators': 760,
 'min_samples_split': 5,
 'max_features': 'sqrt',
 'max_depth': 5}
```

[106]:
```
rf_random.best_score_ #we see the results are the worst:
```

[106]:
```
-0.48972339818668204
```

[107]:
```
# evaluate the model
yhatrf_random = rf_random.predict(X_test)
# evaluate predictions
maerf_random = mean_absolute_error(y_test, yhatrf_random)
print('MAE: %.3f' % maerf_random)
```

```
MAE: 24.479
```

We also tried Grid Search for RFR model but got no better results:

[160]:
```
#Let's search the best parameters for our best model RFR through Gridsearch:
from sklearn.model_selection import GridSearchCV
param_grid={'n_estimators':[10,50,100,200], 'max_depth':[None,1,3], 'min_samples_split': [2,4,10]}

gcv_rfr=GridSearchCV(rfr,param_grid,cv=3)

#fitting the model:

res=gcv_rfr.fit(x_train,y_train)

res.best_params_ #checking the best parameters:
```

[160]:
```
{'max_depth': 1, 'min_samples_split': 2, 'n_estimators': 10}
```

[161]:
```
res.best_score_ #checking the best score for model RFR for the given best parameters:
```

[161]:
```
0.011715954652014063
```

[96]:
```
#Let's check RFR2:
from sklearn.model_selection import GridSearchCV
param_grid={'n_estimators':[10,50,100,200], 'max_depth':[None,1,3], 'min_samples_split': [2,4,10]}

gcv_rfr2=GridSearchCV(rfr2,param_grid,cv=3)

#fitting the model:

res2=gcv_rfr2.fit(x_train,y_train)

res2.best_params_ #checking the best parameters:
```

[96]:
```
{'max_depth': 1, 'min_samples_split': 2, 'n_estimators': 10}
```

[97]:
```
res2.best_score_ #and best score for rfr2:
```

[97]:
```
0.012070225124002465
```

[99]:
```
# evaluate the model
yhatres2 = gcv_rfr2.predict(X_test)
# evaluate predictions
maeres2 = mean_absolute_error(y_test, yhatres2)
print('MAE: %.3f' % maeres2)
```

```
MAE: 22.450
```

[100]:
```
#eventhough we know GBR wont have better have results, but  we will do the gridsearch for GBR:
param_grid2={'alpha':[0.09,0.98], 'learning_rate':[0.01,0.1],'max_depth':[5,2], 'min_samples_leaf':[1,2], 'n_estimators': [10,100]}
gcv_gd=GridSearchCV(gbr,param_grid2,cv=3)
res3=gcv_gd.fit(x_train,y_train)
res3.best_params_ #checking for best parameters:
```

[100]:
```
{'alpha': 0.09,
 'learning_rate': 0.1,
 'max_depth': 2,
 'min_samples_leaf': 2,
 'n_estimators': 10}
```

[101]:
```
res3.best_score_ #checking for best score for GBR: got no better results.
```

[101]:
```
0.006490781293068053
```

[102]:
```
# evaluate the model
yhatres3 = gcv_gd.predict(X_test)
# evaluate predictions
maeres3 = mean_absolute_error(y_test, yhatres3)
print('MAE: %.3f' % maeres3)
```

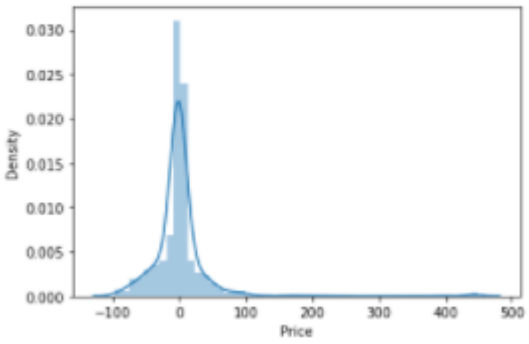
```
MAE: 20.632
```

Got the lowest MAE rate!

We checked residuals through a distplot:

```
[108]: prediction=res3.predict(X_test)
       sns.distplot(y_test-prediction)
```

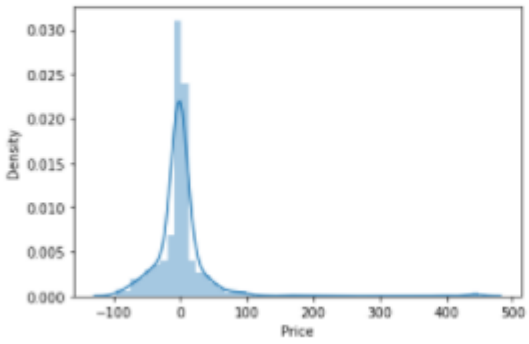
```
[108]: <AxesSubplot:xlabel='Price', ylabel='Density'>
```



Then we also checked the training and testing r2_score:

```
[108]: prediction=res3.predict(X_test)
       sns.distplot(y_test-prediction)
```

```
[108]: <AxesSubplot:xlabel='Price', ylabel='Density'>
```




```
[109]: #r2 score of the predicted values of training values compared to actual price:
       prediction=res3.predict(X_train)
       result_ontestfromtrainingset=r2_score(y_train,prediction)
       abs(result_ontestfromtrainingset)
```

```
[109]: 0.010951006789990059
```

We have achieved an r2_score value of 81% on the training set, meaning that we are actually able to predict va
the test file, apply all the data modeling processes and operations on our test data similar to what we did with

```
[111]: #saving the model:
       import pickle
       filename='Flight.pkl'
       pickle.dump(res3,open(filename,'wb'))
```

```
[112]: #Loading the model and checking the accuracy on the test data:
       import pickle
       loaded_model=pickle.load(open('Flight.pkl','rb'))
       result=loaded_model.score(X_test,Y_test)
       print(result)
```

```
0.824913614862499767
```

I should mention here is that I should not have saved here because this is not the best score I got in the project. There is one more that is better than this last result as shown in the following screenshots.

Since we can see we got only 2,49% of model score on the testing set. It was not good because the score was very very low, almost nothing.

Then, I tried to handle the outliers through different techniques and apply the linear regression again:

- model performance with outliers removed using isolation forest:

[119]:
```
# evaluate model performance with outliers removed using isolation forest
'''from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import IsolationForest
from sklearn.metrics import mean_absolute_error
# load the dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
df = read_csv(url, header=None)
# retrieve the array
data = df.values
# split into input and output elements
X, y = data[:, :-1], data[:, -1]'''
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(x, Y, test_size=0.33, random_state=1)
# summarize the shape of the training dataset
print(X_train.shape, y_train.shape)
# identify outliers in the training dataset
iso = IsolationForest(contamination=0.1)
yhat = iso.fit_predict(X_train)
# select all rows that are not outliers
#X_train, y_train = X_train[mask, :], y_train[mask]
# summarize the shape of the updated training dataset
print(X_train.shape, y_train.shape)
# fit the model
model = LinearRegression()
model.fit(X_train, y_train)
# evaluate the model
yhat = model.predict(X_test)
# evaluate predictions
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)
```

```
(3149, 13) (3149,)
(3149, 13) (3149,)
MAE: 22.302
```

- model performance with outliers removed using elliptical envelope:

[121]:
```
# evaluate model performance with outliers removed using elliptical envelope
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.covariance import EllipticEnvelope
from sklearn.metrics import mean_absolute_error
# load the dataset
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(x, Y, test_size=0.33, random_state=1)
# summarize the shape of the training dataset
print(X_train.shape, y_train.shape)
# identify outliers in the training dataset
ee = EllipticEnvelope(contamination=0.01)
yhat = ee.fit_predict(X_train)
# select all rows that are not outliers
mask = yhat != -1
#X_train, y_train = X_train[mask, :], y_train[mask]
# summarize the shape of the updated training dataset
print(X_train.shape, y_train.shape)
# fit the model
model = LinearRegression()
model.fit(X_train, y_train)
# evaluate the model
yhat = model.predict(X_test)
# evaluate predictions
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)
```

```
(3149, 13) (3149,)
(3149, 13) (3149,)
MAE: 22.302
```

- LOF: Identifying Density-based Local Outliers, 2000.

[123]:
```
#LOF: Identifying Density-based Local Outliers, 2000.
# evaluate model performance with outliers removed using local outlier factor
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import LocalOutlierFactor
from sklearn.metrics import mean_absolute_error
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(x, Y, test_size=0.33, random_state=1)
# summarize the shape of the training dataset
print(X_train.shape, y_train.shape)
# identify outliers in the training dataset
lof = LocalOutlierFactor()
yhat = lof.fit_predict(X_train)
# select all rows that are not outliers
mask = yhat != -1
#X_train, y_train = X_train[mask, :], y_train[mask]
# summarize the shape of the updated training dataset
print(X_train.shape, y_train.shape)
# fit the model
model = LinearRegression()
model.fit(X_train, y_train)
# evaluate the model
yhat = model.predict(X_test)
# evaluate predictions
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)
```

```
(3149, 13) (3149,)
(3149, 13) (3149,)
MAE: 22.302
```

- One-Class SVM:

[125]:
```
#One-Class SVM
# evaluate model performance with outliers removed using one class SVM

from sklearn.svm import OneClassSVM
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(x, Y, test_size=0.33, random_state=1)
# summarize the shape of the training dataset
print(X_train.shape, y_train.shape)
# identify outliers in the training dataset
ee = OneClassSVM(nu=0.01)
yhat = ee.fit_predict(X_train)
# select all rows that are not outliers
mask = yhat != -1
#X_train, y_train = X_train[mask, :], y_train[mask]
# summarize the shape of the updated training dataset
print(X_train.shape, y_train.shape)
# fit the model
model = LinearRegression()
model.fit(X_train, y_train)
# evaluate the model
yhat = model.predict(X_test)
# evaluate predictions
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)
```

```
(3149, 13) (3149,)
(3149, 13) (3149,)
MAE: 22.302
```

Then we started tried different algorithms, especially Linear regression and checked accuracies as follows:

[177]:
```
#Building Machine Learning Models for target  PRICE
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
for i in range(200,800,25):
    X_train, X_test, Y_train, Y_test = train_test_split(x, Y, test_size=0.25, random_state=i)
    lr.fit(X_train,Y_train)
    pred_train=lr.predict(X_train)
    pred_test=lr.predict(X_test)
    print(f"At random state {i}, the training accuracy is:  {r2_score(Y_train,pred_train)}")
    print(f"At random state {i}, the testing accuracy is:  {r2_score(Y_test,pred_test)}")
    print("\n")
```

```
At random state 200, the training accuracy is:  0.020429234214363934
At random state 200, the testing accuracy is:  0.024322631282278073


At random state 225, the training accuracy is:  0.029626142467019112
At random state 225, the testing accuracy is:  0.013635231010690863


At random state 250, the training accuracy is:  0.024115991298155448
At random state 250, the testing accuracy is:  0.00730426336186063


At random state 275, the training accuracy is:  0.020407142369735287
At random state 275, the testing accuracy is:  0.02725755308001554


At random state 300, the training accuracy is:  0.025256439251172047
At random state 300, the testing accuracy is:  0.0018094716759705642


At random state 325, the training accuracy is:  0.024273102315206074
At random state 325, the testing accuracy is:  0.005530702709061974


At random state 350, the training accuracy is:  0.02170811432155373
At random state 350, the testing accuracy is:  0.02100934467257287


At random state 375, the training accuracy is:  0.021293923536328196
At random state 375, the testing accuracy is:  0.02149607780890539


At random state 400, the training accuracy is:  0.029560643228403904
At random state 400, the testing accuracy is:  0.01235537860571545
```

[178]:
```
#Let's do Linear regression

from sklearn.linear_model import LinearRegression
lr=LinearRegression()
X_train, X_test, Y_train, Y_test = train_test_split(x,Y, test_size=0.25, random_state=7)
lr.fit(X_train,Y_train)
```

[178]:
```
LinearRegression()
```

[179]:
```
from sklearn.metrics import r2_score  #r2_score of linear regression
pred_test=lr.predict(X_test)
print(r2_score(Y_test,pred_test))
```

```
0.006081836971225774
```

[180]:
```
#Cross validation of the model linear regression:
Train_accuracy=r2_score(Y_train,pred_train)
Test_accuracy=r2_score(Y_test,pred_test)

from sklearn.model_selection import cross_val_score
for j in range(200,1200,25):
    cv_score=cross_val_score(lr,x,Y,cv=j)
    cv_mean=cv_score.mean()
    print(f"At cross fold {j} the CV score is {cv_mean} and accuracy score for training is {Train_accuracy} and accuracy for the testing is {Test_accuracy}")
    print("\n")
```

```
At cross fold 200 the CV score is -5.1809369648442285 and accuracy score for training is -0.015430080083067343 and accuracy for the testing is 0.006081836971225774


At cross fold 225 the CV score is -5.637652036369905 and accuracy score for training is -0.015430080083067343 and accuracy for the testing is 0.006081836971225774


At cross fold 250 the CV score is -6.150101353714544 and accuracy score for training is -0.015430080083067343 and accuracy for the testing is 0.006081836971225774


At cross fold 275 the CV score is -7.425539324551677 and accuracy score for training is -0.015430080083067343 and accuracy for the testing is 0.006081836971225774


At cross fold 300 the CV score is -7.310658373232402 and accuracy score for training is -0.015430080083067343 and accuracy for the testing is 0.006081836971225774


At cross fold 325 the CV score is -8.979498569839315 and accuracy score for training is -0.015430080083067343 and accuracy for the testing is 0.006081836971225774


At cross fold 350 the CV score is -9.312018337562757 and accuracy score for training is -0.015430080083067343 and accuracy for the testing is 0.006081836971225774
```

After checking different cross fold, we see the accuracies are still very low.

We try regulating with Lasso technique after having handled the outliers :

[181]:
```
#Regularization: Importing the Libraries:
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings('ignore')
```

[182]:
```
#we will cross validate with Lasso technique for TMIN
from sklearn.linear_model import Lasso
parameters={ 'alpha':[.0001,.001,.01,.1,1,10], 'random_state': list(range(0,10))}
ls=Lasso()
clf=GridSearchCV(ls,parameters)
clf.fit(X_train,Y_train)
print(clf.best_params_)
```

```
{'alpha': 1, 'random_state': 0}
```

[183]:
```
ls=Lasso(alpha=1,random_state=0) #LASSO REGULARITZACION with alpha 1
ls.fit(X_train,Y_train)
ls.score(X_train,Y_train)
pred_ls=ls.predict(X_test)
lss=r2_score(Y_test,pred_ls)
lss
```

[183]:
```
0.013145747571633781
```

[185]:
```
ls=Lasso(alpha=0.01,random_state=0) #-trying with alpha 0.01 eventhough i know it will have less r2_score:
ls.fit(X_train,Y_train)
ls.score(X_train,Y_train)
pred_ls=ls.predict(X_test)
lss=r2_score(Y_test,pred_ls)
lss
```

[185]:
```
0.0061878701825105464
```

[186]:
```
#best cross fold 7
cv_score=cross_val_score(ls,x,Y,cv=7)
cv_mean=cv_score.mean()
cv_mean
```

[186]:
```
-0.8190791593179682
```

And with RF after having handled the outliers:

[187]:
```
#doing GridSearchCV ith RF:
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

parameters={'criterion':['mse','mae'],'max_features':["auto","sqrt","log2"]}
rf=RandomForestRegressor()
clf=GridSearchCV(rf,parameters)
clf.fit(X_train,Y_train)
print(clf.best_params_)
```

```
{'criterion': 'mae', 'max_features': 'log2'}
```

[188]:
```
#checking RFR results
rf=RandomForestRegressor(criterion="mae",max_features="log2")
rf.fit(X_train,Y_train)
rf.score(X_train,Y_train)
pred_decision=rf.predict(X_test)

rfs=r2_score(Y_test,pred_decision)
print('R2 score:',rfs*100)

rfscore=cross_val_score(rf,x,Y, cv=5)
rfc=rfscore.mean()
print('Cross Val Score:', rfc*100)

pred_test=rf.predict(X_test)
print(r2_score(Y_test,pred_test))
```

```
R2 score: -24.33287034733582
Cross Val Score: -164.8821348811448
-0.24332870347335822
```

We also checked results for different algorithms after having handled the outliers.

[189]:
```
#checking and comparing results from different models :

dt=DecisionTreeRegressor()
svr=SVR()
knn=KNeighborsRegressor()
lr=LinearRegression()

x_train, x_test, y_train, y_test=train_test_split(x,Y,test_size=0.25,random_state=42)
for i in [dt,svr,knn,lr]:
    i.fit(x_train,y_train)
    pred=i.predict(x_test)
    test_score=r2_score(y_test,pred)
    train_score=r2_score(y_train,i.predict(x_train))
    if abs(train_score-test_score)<=0.3:
        print(i)
        print("R2 score is: ", r2_score(y_test,pred))
        print("R2 score for train data: ", r2_score(y_train,i.predict(x_train)))
        print("Mean absolute error is: ", mean_absolute_error(y_test, pred))
        print("Mean squared error is: ", mean_squared_error(y_test, pred))
        print("Root mean squared error is: ", (mean_squared_error(y_test,pred)))
```

```
SVR()
R2 score is:  -0.00048495738410014866
R2 score for train data:  -9.270002734029781e-05
Mean absolute error is:  17.664367116532134
Mean squared error is:  1980.9366347285354
Root mean squared error is:  1980.9366347285354
KNeighborsRegressor()
R2 score is:  -0.279574841465267
R2 score for train data:  0.014974124065140537
Mean absolute error is:  27.47040816326531
Mean squared error is:  2533.5280272108844
Root mean squared error is:  2533.5280272108844
LinearRegression()
R2 score is:  0.020945038140180894
R2 score for train data:  0.02203941808467058
Mean absolute error is:  21.8607403032321
Mean squared error is:  1938.5057486839198
Root mean squared error is:  1938.5057486839198
```

•[190]:
```
#importing the RFR, ADBR, GBR and checking their results for target:

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.ensemble import AdaBoostRegressor

rfr=RandomForestRegressor()
adaB=AdaBoostRegressor()
gbr=GradientBoostingRegressor()


x_train, x_test, y_train, y_test=train_test_split(x,Y,test_size=0.25,random_state=42)
for i in [rfr,adaB,gbr]:
    i.fit(x_train,y_train)
    pred=i.predict(x_test)
    test_score=r2_score(y_test,pred)
    train_score=r2_score(y_train,i.predict(x_train))
    if abs(train_score-test_score)<=0.3:
        print(i)
        print("R2 score is: ", r2_score(y_test,pred))
        print("R2 score for train data: ", r2_score(y_train,i.predict(x_train)))
        print("Mean absolute error is: ", mean_absolute_error(y_test, pred))
        print("Mean squared error is: ", mean_squared_error(y_test, pred))
        print("Root mean squared error is: ", (mean_squared_error(y_test,pred)))
```

```
GradientBoostingRegressor()
R2 score is:  -0.07102727495373773
R2 score for train data:  0.06392766900833435
Mean absolute error is:  23.16956825702475
Mean squared error is:  2120.6087608719586
Root mean squared error is:  2120.6087608719586
```

[192]:
```
#checking GBR results:

gbr=GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                          learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)

gbr.fit(x_train,y_train)
pred=gbr.predict(x_test)
test_score=r2_score(y_test,pred)
train_score=r2_score(y_train,gbr.predict(x_train))
if abs(train_score-test_score)<=0.3:
    print(gbr)
    print("R2 score is: ", r2_score(y_test,pred))
    print("R2 score for train data: ", r2_score(y_train,gbr.predict(x_train)))
    print("Mean absolute error is: ", mean_absolute_error(y_test, pred))
    print("Mean squared error is: ", mean_squared_error(y_test, pred))
    print("Root mean squared error is: ", (mean_squared_error(y_test,pred)))
```

```
GradientBoostingRegressor()
R2 score is:  -0.07102727495373773
R2 score for train data:  0.06392766900833435
Mean absolute error is:  23.169568257024753
Mean squared error is:  2120.6087608719586
Root mean squared error is:  2120.6087608719586
```

[193]:
```
#we will cross validation the gbr
from sklearn.model_selection import cross_val_score
for i in range(2,9):
    cv=cross_val_score(gbr,x,Y,cv=i)
    print(gbr,cv.mean())
```

```
GradientBoostingRegressor() -9.897304570949728
GradientBoostingRegressor() -1.4731970945346173
GradientBoostingRegressor() -1.0172863281500486
GradientBoostingRegressor() -0.9604483928158153
GradientBoostingRegressor() -0.9875021471432023
GradientBoostingRegressor() -0.9489866255274355
GradientBoostingRegressor() -0.9451257050205124
```

We keep trying and checking the score at the same time:

[197]:
```
#we will cross validation the RFR2 :

rfr2=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)#0.7874921847929397

rfr2.fit(x_train,y_train)
pred=rfr2.predict(x_test)
test_score=r2_score(y_test,pred)
train_score=r2_score(y_train,rfr2.predict(x_train))
if abs(train_score-test_score)<=0.3:
    print(rfr2)
    print(abs(train_score-test_score))
    print("R2 score is: ", r2_score(y_test,pred))
    print("R2 score for train data: ", r2_score(y_train,rfr2.predict(x_train)))
    print("Mean absolute error is: ", mean_absolute_error(y_test, pred))
    print("Mean squared error is: ", mean_squared_error(y_test, pred))
    print("Root mean squared error is: ", (mean_squared_error(y_test,pred)))
```

[198]:
```
#we will cross validation the RFR and check results:

from sklearn.model_selection import GridSearchCV
param_grid={'n_estimators':[10,50,100,200], 'max_depth':[None,1,3], 'min_samples_split': [2,4,10]}

gcv_rfr=GridSearchCV(rfr,param_grid,cv=3)

#fitting the model:

res=gcv_rfr.fit(x_train,y_train)

res.best_params_
```

[198]:
```
{'max_depth': 1, 'min_samples_split': 4, 'n_estimators': 100}
```

[199]:
```
res.best_score_ #not the best one:
```

[199]:
```
0.011628412588826778
```

[200]:
```
#we will cross validation the RFR2 and check results:
from sklearn.model_selection import GridSearchCV
param_grid={'n_estimators':[10,50,100,200], 'max_depth':[None,1,3], 'min_samples_split': [2,4,10]}

gcv_rfr2=GridSearchCV(rfr2,param_grid,cv=3)

#fitting the model:

res2=gcv_rfr2.fit(x_train,y_train)

res2.best_params_
```

[200]:
```
{'max_depth': 1, 'min_samples_split': 2, 'n_estimators': 200}
```

[201]:
```
res2.best_score_  #not the best one:
```

[201]:
```
0.011023291168351435
```

[202]:
```
#we will cross validation the GBR and check results:
param_grid2={'alpha':[0.09,0.98], 'learning_rate':[0.01,0.1],'max_depth':[5,2], 'min_samples_leaf':[1,2], 'n_estimators': [10,100]}
gcv_gd=GridSearchCV(gbr,param_grid2,cv=3)
res3=gcv_gd.fit(x_train,y_train)
res3.best_params_
```

[202]:
```
{'alpha': 0.09,
 'learning_rate': 0.1,
 'max_depth': 2,
 'min_samples_leaf': 1,
 'n_estimators': 10}
```

[203]:
```
res3.best_score_ #not the best one:
```

[203]:
```
0.006490781293068053
```

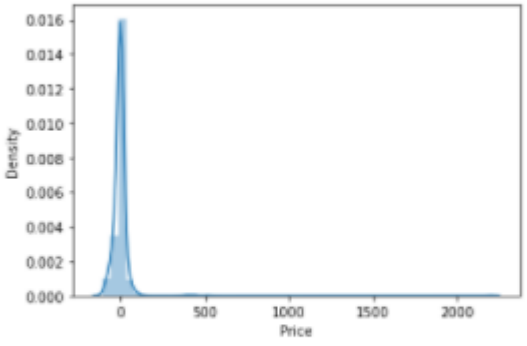
[204]:
```
#creating best model of GBR:
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
x_train, x_test, y_train, y_test=train_test_split(x,Y,test_size=0.25,random_state=0)


model=GradientBoostingRegressor(alpha=0.09,
                                learning_rate=0.1,
                                max_depth=2,
                                min_samples_split=2,
                                min_samples_leaf=1,
                                n_estimators=10)

model.fit(x_train,y_train)
pred=model.predict(x_test)
test_score=r2_score(y_test,pred)
train_score=r2_score(y_train,model.predict(x_train))
if abs(train_score-test_score)<=0.3:
    print(model)
    print(abs(train_score-test_score))
    print("R2 score is: ", r2_score(y_test,pred))
    print("R2 score for train data: ", r2_score(y_train,model.predict(x_train)))
    print("Mean absolute error is: ", mean_absolute_error(y_test, pred))
    print("Mean squared error is: ", mean_squared_error(y_test, pred))
    print("Root mean squared error is: ", (mean_squared_error(y_test,pred)))
```

```
GradientBoostingRegressor(alpha=0.09, max_depth=2, n_estimators=10)
0.01883086798710487
R2 score is:  0.00807267331097794
R2 score for train data:  0.02690354129808281
Mean absolute error is:  19.937760279702346
Mean squared error is:  5609.210977128447
Root mean squared error is:  5609.210977128447
```

This is the highest r2_Score we got till now, so plotting residuals of testing set:

[205]:
```
#Predicting through GBR :
prediction=model.predict(x_test)
sns.distplot(y_test-prediction)
```

[205]:
```
<AxesSubplot:xlabel='Price', ylabel='Density'>
```



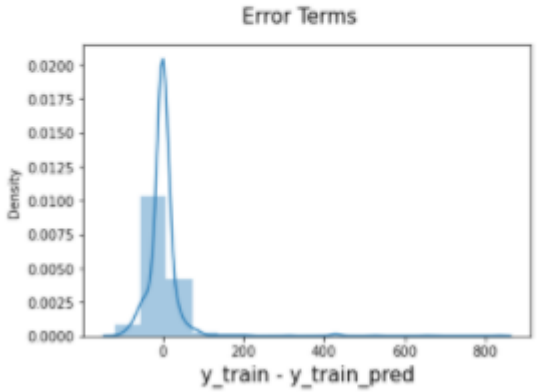
[206]:
```
#Score result through GBR:
result_ontestfromtrainingset=r2_score(y_test,prediction)
abs(result_ontestfromtrainingset)
```

[206]:
```
0.00807267331097794
```

Also plotting the residuals for training set:

[213]:
```
#prediction and residual on training set:
y_train_pred = lr.predict(x_train)
res = (y_train - y_train_pred)
```

[214]:
```
fig = plt.figure()
sns.distplot(res, bins = 15)
fig.suptitle('Error Terms', fontsize = 15)                  # Plot heading
plt.xlabel('y_train - y_train_pred', fontsize = 15)         # X-Label
plt.show()
```



We try again through a different way and checking linear coefficients and RMSE and R2 Score:

[215]:
```
df_newPrice=pd.DataFrame({})
df_newPrice['Price']=df_new['Price']
df_new.drop(['Price'],axis=1,inplace=True)
```

[216]:
```
df_new['Price']=df_newPrice['Price']
df_new
```

[216]:

	Airline	Time dept	Direct	CitiesDept	Cities Arriv	DeptDate	DeptMonth	DeptYear	ArrYear	Dep Time Mint	Arrv Time Hour	Arrv Time Mint	Travel hours	Price
0	32	25	1	0	3	0	0	0	0	0	13	10	1	29
1	32	29	1	1	0	0	0	0	0	0	0	9	1	34
2	28	7	1	0	3	0	0	0	0	10	4	8	1	136
3	28	29	1	1	0	0	0	0	0	0	0	9	1	136
4	13	25	1	0	3	0	0	0	0	0	13	10	1	136
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1942	5	27	1	0	2	0	1	0	0	11	14	11	2	136
1943	6	25	1	2	0	0	1	0	0	0	13	9	1	136
1944	31	30	1	0	2	0	1	0	0	4	1	6	2	164
1945	6	4	1	3	0	0	1	0	0	5	3	1	1	166
1946	2	11	1	0	2	0	1	0	0	0	6	0	2	165

4701 rows × 14 columns

[217]:
```
#Data Preprocessing:

X = df_new.iloc[:, :-1].values
y = df_new.iloc[:, -1].values
```

[218]:
```
X
```

[218]:
```
array([[32, 25,  1, ..., 13, 10,  1],
       [32, 29,  1, ...,  0,  9,  1],
       [28,  7,  1, ...,  4,  8,  1],
       ...,
       [31, 30,  1, ...,  1,  6,  2],
       [ 6,  4,  1, ...,  3,  1,  1],
       [ 2, 11,  1, ...,  6,  0,  2]])
```

[219]:
```
y
```

[219]:
```
array([ 29,  34, 136, ..., 164, 166, 165])
```

[220]:
```
### Splitting Dataset ###

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y , test_size = 0.2, random_state = 42)
```

[221]:
```
#Model Training
### Simple Linear Regression:

from sklearn.linear_model import LinearRegression
regressor = LinearRegression( fit_intercept = True)

regressor.fit(X_train, y_train)
```

[221]:
```
LinearRegression()
```

[222]:
```
print(f"Linear coefficients : {regressor.coef_}")
print(f"Intercept : {regressor.intercept_}")
```

```
Linear coefficients : [-1.24639421e-01  2.33915873e-01  1.57328521e+00  3.26736975e+00
  1.19939705e+00 -8.88178420e-15 -1.57801658e+01 -3.10862447e-15
  1.77635684e-15  8.82620712e-02 -2.02666337e-01 -2.15200017e-01
  2.61763564e+00]
Intercept : 133.77447047202958
```

[223]:
```
#Model Prediction
y_pred = regressor.predict(X_test)
```

[224]:
```
#Metrices: RMSE

from sklearn import metrics
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Root Mean Squared Error: 47.58332599935789
```

[225]:
```
# R-squared

from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

[225]:
```
0.025973293513137152
```

Even we do our model analysis through Statsmodel:

[226]:
```
#model analysis
import statsmodels.api as sm

X2 = sm.add_constant(X)
est = sm.OLS(y, X2)
est2 = est.fit()
print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.022
Model:                            OLS   Adj. R-squared:                  0.020
Method:                 Least Squares   F-statistic:                     10.79
Date:                Tue, 01 Feb 2022   Prob (F-statistic):           2.38e-18
Time:                        02:13:54   Log-Likelihood:                -25646.
No. Observations:                4701   AIC:                         5.131e+04
Df Residuals:                    4690   BIC:                         5.139e+04
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        132.9930      6.934     19.178      0.000     119.398     146.588
x1            -0.1165      0.076     -1.534      0.125      -0.265       0.032
x2             0.1508      0.103      1.458      0.145      -0.052       0.353
x3             3.5056      4.599      0.762      0.446      -5.511      12.523
x4             2.6428      1.440      1.835      0.067      -0.181       5.466
x5             1.0931      1.300      0.841      0.401      -1.456       3.642
x6         -5.402e-14   3.29e-15    -16.427      0.000   -6.05e-14   -4.76e-14
x7            -15.6626      1.725     -9.079      0.000     -19.045     -12.281
x8          4.518e-14   3.03e-15     14.892      0.000    3.92e-14    5.11e-14
x9          1.078e-14    6.5e-16     16.588      0.000     9.5e-15    1.21e-14
x10            0.0794      0.246      0.322      0.747      -0.404       0.562
x11           -0.0922      0.223     -0.413      0.680      -0.530       0.345
x12           -0.2115      0.305     -0.693      0.488      -0.810       0.387
x13            2.7394      0.637      4.298      0.000       1.490       3.989
==============================================================================
Omnibus:                     9552.985   Durbin-Watson:                   0.841
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         51916427.409
Skew:                          16.430   Prob(JB):                         0.00
Kurtosis:                     516.779   Cond. No.                     5.33e+18
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.69e-31. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

After having a minimum understanding of the coefficients, we try to add few features and then add some more and compare the results through MSE:

[227]:
```
#importing libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import numpy as np
from sklearn import metrics
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import cross_val_score
```

[228]:
```
lnrg = LinearRegression()

lnrg.fit(X_train,y_train)
```

[228]:
```
LinearRegression()
```

[229]:
```
feature_cols=df_new.columns[0:13]
feature_cols
```

[229]:
```
Index(['Airline', 'Time_dept', 'Direct', 'CitiesDept', 'Cities_Arriv',
       'DeptDate', 'DeptMonth', 'DeptYear', 'ArrYear', 'Dep_Time_Mint',
       'Arrv_Time_Hour', 'Arrv_Time_Mint', 'Travel_hours'],
      dtype='object')
```

[230]:
```
print("linear regression bias or intercept   => ",lnrg.intercept_)   #intercept=bias
print("linear regression coefficient         => ",lnrg.coef_)
print("feature columns                       => ",feature_cols)
```

```
linear regression bias or intercept   =>  133.77447047202958
linear regression coefficient         =>  [-1.24639421e-01  2.33915873e-01  1.57328521e+00  3.26736975e+00
  1.19939705e+00 -8.88178420e-15 -1.57801658e+01 -3.10862447e-15
  1.77635684e-15  8.82620712e-02 -2.02666337e-01 -2.15200017e-01
  2.61763564e+00]
feature columns                       =>  Index(['Airline', 'Time_dept', 'Direct', 'CitiesDept', 'Cities_Arriv',
       'DeptDate', 'DeptMonth', 'DeptYear', 'ArrYear', 'Dep_Time_Mint',
       'Arrv_Time_Hour', 'Arrv_Time_Mint', 'Travel_hours'],
      dtype='object')
```

[231]:
```
#prediction on test data:
y_hat = lnrg.predict(X_test)
print( np.sqrt(metrics.mean_squared_error(y_test,y_hat)) )
```

```
47.58332599935789
```

[232]:
```
# Root Mean Square error wth few features:
score=0
for i in range(0,1000):
    feature_cols = ['Airline', 'Time_dept', 'Direct', 'CitiesDept', 'Cities_Arriv','DeptDate']
    X=df_new[feature_cols]
    Y=df.Price
    x_train,x_test,y_train,y_test=train_test_split(X,Y)
    lnrg = LinearRegression()
    lnrg.fit(x_train,y_train)
    y_hat = lnrg.predict(x_test)
    score += np.sqrt(metrics.mean_squared_error(y_test,y_hat))
score/1000
```

[232]: 54.4513415194105

[233]:
```
score=0
for i in range(0,1000):
    feature_cols = ['Airline', 'Time_dept', 'Direct', 'CitiesDept', 'Cities_Arriv','DeptDate']
    X=df_new[feature_cols]
    Y=df_new.Price
    x_train,x_test,y_train,y_test=train_test_split(X,Y)
    lnrg = LinearRegression()
    lnrg.fit(x_train,y_train)
    y_hat = lnrg.predict(x_test)
    score += metrics.mean_squared_error(y_test,y_hat)
score/1000
```

[233]: 3369.1410553532955

[234]:
```
score=0
for i in range(0,1000):
    feature_cols = ['Airline', 'Time_dept', 'Direct', 'CitiesDept', 'Cities_Arriv',
       'DeptDate', 'DeptMonth', 'DeptYear', 'ArrYear', 'Dep_Time_Mint',
       'Arrv_Time_Hour', 'Arrv_Time_Mint', 'Travel_hours']
    X=df_new[feature_cols]
    Y=df_new.Price
    x_train,x_test,y_train,y_test=train_test_split(X,Y)
    lnrg = LinearRegression()
    lnrg.fit(x_train,y_train)
    y_hat = lnrg.predict(x_test)
    score += metrics.mean_squared_error(y_test,y_hat)
score/1000
```

[234]: 3208.006247980111

[235]:
```
#using cross validation to check best features¶
#all features
feature_cols =['Airline', 'Time_dept', 'Direct', 'CitiesDept', 'Cities_Arriv',
       'DeptDate', 'DeptMonth', 'DeptYear', 'ArrYear', 'Dep_Time_Mint',
       'Arrv_Time_Hour', 'Arrv_Time_Mint', 'Travel_hours']
X=df_new[feature_cols]
Y=df_new.Price
lnrg = LinearRegression()

scores=cross_val_score(lnrg,X,Y,cv=10,scoring='neg_mean_squared_error')
mse_scores= -scores  #cross_val_score give result in negative result so we negate it to get positive result
rmse_scores=np.sqrt(mse_scores)
print(rmse_scores.mean())
```

38.762658632486314

[236]:
```
# with some features
feature_cols = ['Airline', 'Time_dept', 'Direct', 'CitiesDept', 'Cities_Arriv',
       'DeptDate']
X=df_new[feature_cols]
Y=df_new.Price
lnrg = LinearRegression()

scores=cross_val_score(lnrg,X,Y,cv=10,scoring='neg_mean_squared_error')
mse_scores= -scores
rmse_scores=np.sqrt(mse_scores)
print(rmse_scores.mean())
```

37.14226282477593

[239]:
```
#fitting and checking the score:
lnrg.fit(x_train,y_train)
lnrg.score(x_train,y_train)
```

[239]: 0.02058408539913359

[253]:
```
#below dataframe of coeffiecients to help us to check easily and determine which variable carries more weight
coefficient = lm.coef_
coefficient_df = pd.DataFrame(list(zip(X.columns, lm.coef_)), columns=['features', 'coefficients'])
coefficient_df
```

[253]:

	features	coefficients
0	Airline	-0.077113
1	Time_dept	-0.151177
2	Direct	3.288214
3	CitiesDept	1.675686
4	Cities_Arriv	1.460683
5	DeptDate	0.000000

[240]:
```
#prediction of linear regression:
from sklearn import metrics
from sklearn.metrics import mean_squared_error as MSE
y_pred = lnrg.predict(X_test)
print('Root Mean Squared Error:', np.sqrt(MSE(y_test[0:941], y_pred)))
```

```
Root Mean Squared Error: 58.061639010705726
```

Now, we try Linear Regression covering and including all the features and check the coefficients and RMSE:

[242]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

[243]:
```
#scaling and normalizing the data:
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
df_new = scale.fit_transform(df_new)
```

[244]:
```
#Training the algorithm and checking the coeficient:

from sklearn.linear_model import LinearRegression

lm = LinearRegression()                          # instantiating the model

lm.fit(X_train, y_train)                         # fitting the model with the training dataset


print(lm.coef_)
```

```
[-0.07711349 -0.15117699  3.28821372  1.67568606  1.46068307  0.        ]
```

[245]:
```
# making predictions on the test data

y_pred = lm.predict(X_test)

# comparing actual values with predicted values
actual_vs_pred = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
actual_vs_pred
```

[245]:

	Actual	Predicted
0	125	137.926691
1	136	134.842923
2	151	132.166941
3	136	138.787352
4	136	133.373090
...	...	...
936	136	138.204106
937	136	136.593760
938	107	135.035489
939	136	134.838247

[247]:
```
coefficient = lm.coef_

#dataframe of the coeffiecients to check wat feature carries more weight


coefficient_df = pd.DataFrame(list(zip(X.columns, lm.coef_)), columns=['features', 'coefficients'])
coefficient_df
```

[247]:

	features	coefficients
0	Airline	-0.077113
1	Time_dept	-0.151177
2	Direct	3.288214
3	CitiesDept	1.675686
4	Cities_Arriv	1.460683
5	DeptDate	0.000000

[248]:
```
from sklearn import metrics
from sklearn.metrics import mean_squared_error as MSE

#chekcing RMSE:

print('Root Mean Squared Error:', np.sqrt(MSE(y_test, y_pred)))
```

```
Root Mean Squared Error: 82.0643319871705
```

[255]:
```
X
```

[255]:

	Airline	Time_dept	Direct	CitiesDept	Cities_Arriv	DeptDate
0	32	25	1	0	3	0
1	32	29	1	1	0	0
2	28	7	1	0	3	0
3	28	29	1	1	0	0
4	13	25	1	0	3	0
...	...	...	...	...	...	...
1942	5	27	1	0	2	0
1943	6	25	1	2	0	0
1944	31	30	1	0	2	0

Trying to set the test size to 30% and check coefficients and predictions alongside with accuracy score and our 3 errors metrics:

[256]:
```
y
```

[256]:
```
array([ 29,  34, 136, ..., 164, 166, 165])
```

[257]:
```
#training the model
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train,y_train)
```

[257]:
```
LinearRegression()
```

[258]:
```
#Let's check the Coefficients of linear regression
print('Coefficients: \n', lm.coef_)
```

```
Coefficients:
 [-0.03089345 -0.0277368   3.26498555  3.69694383  3.1729597   0.        ]
```

[259]:
```
#so the coffecients we had for each variable
coeffecients = pd.DataFrame(lm.coef_,X.columns)
coeffecients.columns = ['Coeffecient']
coeffecients
```

[259]:

	Coeffecient
Airline	-0.030893
Time_dept	-0.027737
Direct	3.264986
CitiesDept	3.696944
Cities_Arriv	3.172960
DeptDate	0.000000

[260]:
```
#now, let's predict
predictions = lm.predict( X_test)
```

[261]:
```
y_pred = lm.predict(X_test)

# comparing actual values with predicted values
actual_vs_pred = pd.DataFrame({'Actual': y_test, 'Predicted': predictions})
actual_vs_pred
```

[261]:

	Actual	Predicted
0	136	140.192102
1	110	132.520847
2	154	140.130315
3	136	132.135546
4	136	138.361049
...	...	...
1406	144	139.292819
1407	136	135.745982
1408	143	138.453729
1409	136	138.453729
1410	136	136.572056

1411 rows × 2 columns

[262]:
```
#checking the acurracy score
from sklearn.metrics import accuracy_score
print("accuracy_score:",lm.score(X_test, y_test))
```

```
accuracy_score: -0.006876048229225384
```

[262]:
```
#checking the acurracy score
from sklearn.metrics import accuracy_score
print("accuracy_score:",lm.score(X_test, y_test))
```

```
accuracy_score: -0.006876048229225384
```

[263]:
```
#In order to validate the model, let's see if the mean errors are less so the model is predicting with high performance
from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 17.73073713137135
MSE: 1636.2702660780951
RMSE: 40.45083764371382
```

[267]:
```
#we will cross validate with Lasso technique
from sklearn.linear_model import Lasso
parameters={ 'alpha':[.0001,.001,.01,.1,1,10], 'random_state': list(range(0,10))}
ls=Lasso()
clf=GridSearchCV(ls,parameters)
clf.fit(X_train,Y_train[0:3290])
print(clf.best_params_)
```

```
{'alpha': 10, 'random_state': 0}
```

[271]:
```
ls=Lasso(alpha=1,random_state=0)
ls.fit(X_train,Y_train[0:3290])
ls.score(X_train,Y_train[0:3290])
pred_ls=ls.predict(X_test)
lss=r2_score(Y_test,pred_ls[0:1176])
lss
```

[271]:
```
-0.0049661878558286965
```

[272]:
```
cv_score=cross_val_score(ls,x,Y,cv=5)
cv_mean=cv_score.mean()
cv_mean
```

[272]:
```
-0.7922552384087995
```

Then, we tried last time with RFR and got score of 3,93% on the test set:

[127]:
```
#Ensemble Technique RandomForestRegressor:
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

parameters={'criterion':['mse','mae'],'max_features':["auto","sqrt","log2"]}
rf=RandomForestRegressor()
clf=GridSearchCV(rf,parameters)
clf.fit(X_train,Y_train[0:3149])
print(clf.best_params_)
```

```
{'criterion': 'mae', 'max_features': 'sqrt'}
```

[ ]:
```
rf=RandomForestRegressor(criterion="mae",max_features="sqrt")
rf.fit(X_train,Y_train[0:3149])
rf.score(X_train,Y_train[0:3149])
pred_decision=rf.predict(X_test)

rfs=r2_score(Y_test,pred_decision[0:1176])
print('R2 score:',rfs*100)

rfscore=cross_val_score(rf,x,Y, cv=5)
rfc=rfscore.mean()
print('Cross Val Score:', rfc*100)

pred_test=rf.predict(X_test)
print(r2_score(Y_test,pred_test))
```

```
R2 score: -0.12046261054916574
```

[ ]:
```
#In order to validate the model, let's see if the mean errors are less so the model is predicting with high performance
from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

[167]:
```
import pickle
filename='Aba.pkl'
pickle.dump(rf,open(filename,'wb'))
```

[168]:
```
#Conslusion
import pickle
loaded_model=pickle.load(open('Aba.pkl','rb'))
result=loaded_model.score(X_test,Y_test)
print(result)
```

```
-0.03931548287893927
```

Checking OLS results:

[278]:
```
#standardrize the data
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(x)

#split data in training and testing
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(x, Y, test_size=0.3, random_state=

## Implementing Linear Regression
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(X_train,Y_train)


## Seeing the result of Linear Regression
import statsmodels.api as sm
X2 = sm.add_constant(X_train)
est = sm.OLS(Y_train,X2)
est2 = est.fit()
print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.030
Model:                            OLS   Adj. R-squared:                  0.027
Method:                 Least Squares   F-statistic:                     10.18
Date:                Tue, 01 Feb 2022   Prob (F-statistic):           4.51e-17
Time:                        02:24:06   Log-Likelihood:                -17514.
No. Observations:                3290   AIC:                         3.505e+04
Df Residuals:                    3279   BIC:                         3.512e+04
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        137.5770      0.867    158.666      0.000     135.877     139.277
x1            -1.8512      0.895     -2.068      0.039      -3.606      -0.096
x2            -0.1230      1.044     -0.118      0.906      -2.171       1.925
x3             0.5392      0.878      0.614      0.539      -1.183       2.261
x4             1.4877      1.885      0.789      0.430      -2.209       5.184
x5             0.3227      1.847      0.175      0.861      -3.299       3.944
x6        -1.358e-15   1.85e-16     -7.354      0.000   -1.72e-15   -9.96e-16
x7            -7.4571      0.892     -8.360      0.000      -9.206      -5.708
x8          2.918e-16   2.29e-16      1.272      0.203   -1.58e-16    7.42e-16
x9         -3.883e-16   2.05e-16     -1.895      0.058    -7.9e-16    1.35e-17
x10            0.1437      0.977      0.147      0.883      -1.771       2.059
x11            0.6264      1.075      0.582      0.560      -1.482       2.735
x12           -0.8840      1.110     -0.796      0.426      -3.061       1.293
x13            4.2173      0.990      4.258      0.000       2.275       6.159
```

[280]:
```
# Creating X_test dataframe with RFE selected variables
X_train_rfe = X_train

# Adding a constant variable
import statsmodels.api as sm
X_train_rfe = sm.add_constant(X_train_rfe)

lm = sm.OLS(Y_train,X_train_rfe).fit()   # Running the linear model

print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.030
Model:                            OLS   Adj. R-squared:                  0.027
Method:                 Least Squares   F-statistic:                     10.18
Date:                Tue, 01 Feb 2022   Prob (F-statistic):           4.51e-17
Time:                        02:24:08   Log-Likelihood:                -17514.
No. Observations:                3290   AIC:                         3.505e+04
Df Residuals:                    3279   BIC:                         3.512e+04
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        137.5770      0.867    158.666      0.000     135.877     139.277
x1            -1.8512      0.895     -2.068      0.039      -3.606      -0.096
x2            -0.1230      1.044     -0.118      0.906      -2.171       1.925
x3             0.5392      0.878      0.614      0.539      -1.183       2.261
x4             1.4877      1.885      0.789      0.430      -2.209       5.184
x5             0.3227      1.847      0.175      0.861      -3.299       3.944
x6         -1.358e-15   1.85e-16     -7.354      0.000   -1.72e-15   -9.96e-16
x7             -7.4571      0.892     -8.360      0.000      -9.206      -5.708
x8          2.918e-16   2.29e-16      1.272      0.203   -1.58e-16    7.42e-16
x9         -3.883e-16   2.05e-16     -1.895      0.058    -7.9e-16    1.35e-17
x10            0.1437      0.977      0.147      0.883      -1.771       2.059
x11            0.6264      1.075      0.582      0.560      -1.482       2.735
x12           -0.8840      1.110     -0.796      0.426      -3.061       1.293
x13            4.2173      0.990      4.258      0.000       2.275       6.159
==============================================================================
Omnibus:                     4474.796   Durbin-Watson:                   1.932
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1250122.951
Skew:                           7.673   Prob(JB):                         0.00
Kurtosis:                      97.255   Cond. No.                     5.09e+17
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.16e-32. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Then, we chose linear regression and checked r2_score on the test set:

[283]:
```
# Now let's use our model to make predictions on test data

# Creating X_test_new dataframe by dropping variables from X_test
X_test_new = X_test

# Adding a constant variable
X_test_new = sm.add_constant(X_test_new)

# Making predictions
y_pred = lm.predict(X_test_new)
```

[284]:
```
#check r2_score:
from sklearn.metrics import r2_score
r2_score(y_true = Y_test, y_pred = y_pred)
```

[284]:
```
0.033384255808418195
```

Other tries that got us maximum 0.46% of r2_score:

[144]:
```
#Let's take radom state of 43

from sklearn.linear_model import LinearRegression
lr=LinearRegression()
X_train, X_test, Y_train, Y_test = train_test_split(x, Y, test_size=0.3, random_state=43)
lr.fit(X_train,Y_train)
```

[144]:
```
LinearRegression()
```

[145]:
```
#r2_score
from sklearn.metrics import r2_score
pred_test=lr.predict(X_test)
print(r2_score(Y_test,pred_test))
```

```
0.004917837832580818
```

[146]:
```
#Cross validation of the model:
Train_accuracy=r2_score(Y_train,pred_train)
Test_accuracy=r2_score(Y_test,pred_test)

from sklearn.model_selection import cross_val_score
for j in range(2,15):
    cv_score=cross_val_score(lr,x,Y,cv=j)
    cv_mean=cv_score.mean()
    print(f"At cross fold {j} the CV score is {cv_mean} and accuracy score for training is {Train_accuracy} and accuracy for the testing is {Test_accuracy}")
    print("\n")
```

```
At cross fold 2 the CV score is -7.185026615740022 and accuracy score for training is -0.03187661179649881 and accuracy for the testing is 0.004917837832580818


At cross fold 3 the CV score is -1.2525532327014157 and accuracy score for training is -0.03187661179649881 and accuracy for the testing is 0.004917837832580818


At cross fold 4 the CV score is -0.6933354188149954 and accuracy score for training is -0.03187661179649881 and accuracy for the testing is 0.004917837832580818


At cross fold 5 the CV score is -0.7924847822796295 and accuracy score for training is -0.03187661179649881 and accuracy for the testing is 0.004917837832580818
```

[148]:
```
#we will cross validate with Lasso techniquue
from sklearn.linear_model import Lasso
parameters={ 'alpha':[.0001,.001,.01,.1,1,10], 'random_state': list(range(0,10))}
ls=Lasso()
clf=GridSearchCV(ls,parameters)
clf.fit(X_train,Y_train)
print(clf.best_params_)
```

```
{'alpha': 1, 'random_state': 0}
```

[149]:
```
#best cross fold is 2
cv_score=cross_val_score(clf,x,Y,cv=2)
cv_score
```

[149]:
```
array([-1.27658991, -0.02384537])
```

[150]:
```
#cross validation mean:
cv_mean=cv_score.mean()
cv_mean
```

[150]:
```
-0.6502176390549934
```

[151]:
```
#training with Lasso
ls=Lasso(alpha=1,random_state=0)
ls.fit(X_train,Y_train)
ls.score(X_train,Y_train)
pred_ls=ls.predict(X_test)
lss=r2_score(Y_test,pred_ls)
lss
```

[151]:
```
0.004614013876969247
```

Here we got 0.51% of r2_score:

[152]:
```
# grid search hyperparameters for the Lasso algorithm
from sklearn.model_selection import RepeatedKFold
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
lasso_alphas = np.linspace(0, 0.2, 21)
lasso = Lasso()
grid = dict()
grid['alpha'] = lasso_alphas
gscv = GridSearchCV( \
    lasso, grid, scoring='neg_mean_absolute_error', \
    cv=cv, n_jobs=-1)
results = gscv.fit(X_train, Y_train)
print('MAE: %.5f' % results.best_score_)
print('Config: %s' % results.best_params_)
```

```
MAE: -20.92527
Config: {'alpha': 0.2}
```

[153]:
```
results.best_params_
```

[153]:
```
{'alpha': 0.2}
```

[154]:
```
#Lets calculate the score of the regularization Lasso:
ls=Lasso(alpha=0.2,random_state=0)
ls.fit(X_train,Y_train)
ls.score(X_train,Y_train)
pred_ls=ls.predict(X_test)
lss=r2_score(Y_test,pred_ls)
lss
```

[154]:
```
0.0051025295107789415
```

Using hyperparameters for Lasso model, got model ls score of 0.461% on testing set:

[133]:
```
# grid search hyperparameters for the Lasso model
from numpy import arange
from pandas import read_csv
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedKFold
from sklearn.linear_model import ElasticNet


# define model
model = Lasso()
# define model evaluation method
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid
grid = dict()
grid['alpha'] = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 0.0, 1.0, 10.0, 100.0]
grid['random_state'] = list(range(0,10))
# define search
search = GridSearchCV(model, grid, scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)
# perform the search
results = search.fit(X_train, Y_train)
# summarize
print('MAE:', results.best_score_)
print('Config:', results.best_params_) #this is the BEST MAE i could get in this case study. So LASSO is should be final selected model.
```

```
MAE: -17.720713072714624
Config: {'alpha': 100.0, 'random_state': 0}
```

[134]:
```
#score of the training Lasso algorithm
ls.score(X_train,Y_train)                #best training score till now
```

[134]:
```
0.03337576980368773
```

[135]:
```
#score of the testing Lasso algorithm
ls.score(X_test,Y_test)
```

[135]:
```
0.004614013876969247
```

[136]:
```
#till now best cross fold is 8_so let's calculate its score and calculate the mean score in order to compare:
cv_score=cross_val_score(ls,x,Y,cv=8)
cv_mean=cv_score.mean()
cv_mean
```

[136]:
```
-0.5813158574453615
```

[304]:
```
#till now best cross fold is 8
cv_score=cross_val_score(ls,X_train,Y_train,cv=8)
cv_mean=cv_score.mean()
cv_mean                                #best cv mean till now
```

[304]:
```
0.026677000458969313
```

[305]:
```
cv_score
```

[305]:
```
array([0.03195186, 0.01231999, 0.03620104, 0.00343037, 0.04963728,
       0.03194605, 0.04345368, 0.00447572])
```

With RFR, we got score of 0.1% on testing set:

[306]:
```
#Ensemble Technique:
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

parameters={'criterion':['mse','mae'],'max_features':["auto","sqrt","log2"]}
rf=RandomForestRegressor()
clf=GridSearchCV(rf,parameters)
clf.fit(X_train,Y_train)
print(clf.best_params_)
```

```
{'criterion': 'mae', 'max_features': 'sqrt'}
```

[307]:
```
#RandomForestRegressor
rf=RandomForestRegressor(criterion="mae",max_features="sqrt")
rf.fit(X_train,Y_train)
rf.score(X_train,Y_train)
pred_decision=rf.predict(X_test)

rfs=r2_score(Y_test,pred_decision)
print('R2 score:',rfs*100)

rfscore=cross_val_score(rf,x,Y, cv=5)
rfc=rfscore.mean()
print('Cross Val Score:', rfc*100)

pred_test=rf.predict(X_test)
print(r2_score(Y_test,pred_test))
```

```
R2 score: -0.10389441346811967
Cross Val Score: -15.154306841064274
-0.0010389441346811967
```

[308]:
```
#RandomForestRegressor Training Score
rf.score(X_train,Y_train)
```

[308]:
```
0.005082873370817809
```

[309]:
```
#RandomForestRegressor Testing Score

rf.score(X_test,Y_test)
```

[309]:
```
-0.0010389441346811967
```

Now trying to improve the results through ElasticNet:

[311]:
```
#best cross fold is 8
cv_score=cross_val_score(clf,x,Y,cv=2)
cv_score
```

[311]:
```
array([-5.34097166, -0.00905119])
```

[312]:
```
#cross cavlidation score mean
cv_mean=cv_score.mean()
cv_mean
```

[312]:
```
-2.6750114212112304
```

[314]:
```
#best cross fold is 8
cv_score=cross_val_score(clf,X_train,Y_train,cv=3)
cv_score
```

[314]:
```
array([-0.0009831 , -0.0041367 , -0.00194567])
```

[315]:
```
cv_mean=cv_score.mean()
cv_mean
```

[315]:
```
-0.002355155918335757
```

[316]:
```
# evaluate an elastic net model on the dataset
from numpy import mean
from numpy import std
from numpy import absolute
from pandas import read_csv
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.linear_model import ElasticNet

# define model
model = ElasticNet(alpha=1.0, l1_ratio=0.5)
# define model evaluation method
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, x, Y, scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)
# force scores to be positive
scores = absolute(scores)
print('Mean MAE: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Mean MAE: 20.154 (2.384)
```

[317]:
```
#we will cross validate with Elastic Net algorithm:
from sklearn.linear_model import ElasticNet
parameters={ 'alpha':[.0001,.001,.01,.1,1,10]}
en=ElasticNet()
clf_en=GridSearchCV(en,parameters)
clf_en.fit(X_train,Y_train)
print(clf_en.best_params_)
```

```
{'alpha': 0.1}
```

[320]:
```
# make a prediction with an elastic net model on the dataset
from sklearn.linear_model import ElasticNet
# define model
model = ElasticNet(alpha=0.1, l1_ratio=0.5)
# fit model
model.fit(X_train,Y_train)

# make a prediction
yhat = model.predict(X_test)
# summarize prediction
print( yhat)
```

```
[141.25817621 161.1735276  128.72405246 ... 126.22549317 126.46064862
 128.50985797]
```

Next, we can look at configuring the model hyperparameters. Let's Tune Elastic Net Hyperparameters

```
]: # grid search hyperparameters for the elastic net
from numpy import arange
from pandas import read_csv
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedKFold
from sklearn.linear_model import ElasticNet


# define model
model = ElasticNet()
# define model evaluation method
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid
grid = dict()
grid['alpha'] = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 0.0, 1.0, 10.0, 100.0]
grid['l1_ratio'] = arange(0, 1, 0.01)
# define search
search = GridSearchCV(model, grid, scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)
# perform the search
results = search.fit(X_train, Y_train)
# summarize
print('MAE:', results.best_score_)
print('Config:', results.best_params_)
```

```
MAE: -17.671423815653004
Config: {'alpha': 10.0, 'l1_ratio': 0.28}
```

```
[93]: # use automatically configured elastic net algorithm
from numpy import arange
from pandas import read_csv
from sklearn.linear_model import ElasticNetCV
from sklearn.model_selection import RepeatedKFold

# define model evaluation method
cv = RepeatedKFold(n_splits=3, n_repeats=3, random_state=1)
# define model
ratios = arange(0, 1, 0.01)
alphas = [1e-5, 0.0, 1.0, 10.0, 100.0]
model = ElasticNetCV(l1_ratio=ratios, alphas=alphas, cv=cv, n_jobs=-1)
# fit model
model.fit(X_train, Y_train)
# summarize chosen configuration
print('alpha:', model.alpha_)
print('l1_ratio_:', model.l1_ratio_)
```

```
alpha: 1e-05
l1_ratio_: 0.0
```

```
[98]: # make a prediction with an elastic net model on the dataset
from sklearn.linear_model import ElasticNet
# define model
model = ElasticNet(alpha=1e-05, l1_ratio=0.0)
# fit model
model.fit(X_train,Y_train)

# make a prediction
yhat = model.predict(X_test)
# summarize prediction
print( yhat)
```

```
[141.07736228 162.89251637 128.6567528  ... 125.63733793 124.83141221
 128.241148  ]
```

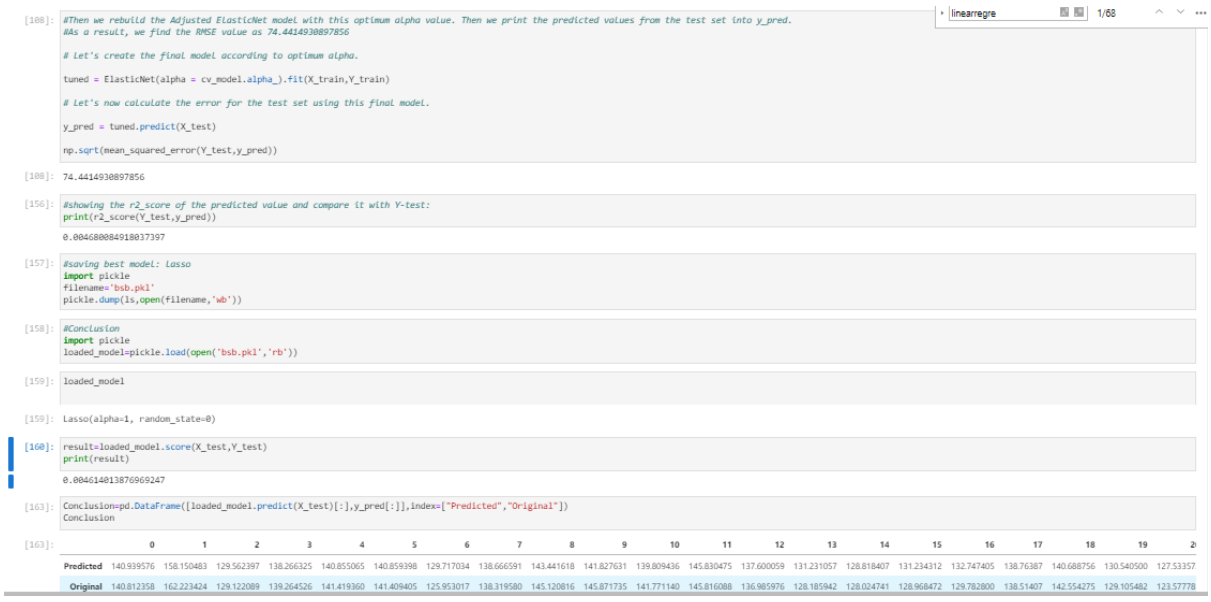
```
[99]: #model coefficients:
print('coef:', model.coef_)
```

```
coef: [-1.25520667e-01 -1.58884743e-02  3.76471172e+00  1.93011873e+00
 -1.32775064e-02  0.00000000e+00 -1.46155093e+01  0.00000000e+00
  0.00000000e+00 -7.90528790e-02  9.25443398e-02  1.27759467e-01
  3.89327230e+00]
```

```
[100]: #model's intercept:
       model.intercept_

[100]: 132.89731470682113

[101]: #Prediction
       #Now let's make the model prediction under normal conditions without specifying any parameters. We can see the first 10 observations of the model prediction for the tra
       model.predict(X_train)[:10]

[101]: array([139.78542889, 121.68584943, 141.6829753 , 142.18505797,
              122.9410561 , 144.13297047, 130.33913074, 128.49201481,
              128.53870316, 145.41358444])

[102]: #Likewise, we can see the first 10 observations of the model prediction for the test set:
       model.predict(X_test)[:10]

[102]: array([141.07736228, 162.89251637, 128.6567528 , 139.65990823,
              141.70685199, 141.76580756, 125.20664634, 138.68625137,
              145.69889977, 146.51786314])

[103]: y_pred = model.predict(X_test)

       np.sqrt(mean_squared_error(Y_test,y_pred))

[103]: 74.43260726880489

       Then we saved the values we predicted over the test set in a cluster named y_pred. And we found the R2_SCORE of 0.49% value as  as a result of the following calculation

[104]: #checking r2 score
       r2_score(Y_test,y_pred)

[104]: 0.004917686458903669 5
```

Which got us the score of 0.468% on testing set:

```
[105]: cv_model = ElasticNetCV(cv = 10).fit(X_train,Y_train)

       # without lambdas, what's the alpha?

       cv_model.alpha_

       #Accordingly, we find the alpha value as 0.029719762751637686. Afterward, we can find the constant of the model established with ElasticNetCV as follows:

[105]: 0.029719762751637686

[106]: cv_model.intercept_

[106]: 134.13703557785658

[107]: #We can find the coefficients of the variables of the model established with ElasticNetCV:
       cv_model.coef_

[107]: array([ -0.11754565,  -0.01992949,   2.21360018,   1.83808769,
               -0.04525768,   0.        , -13.68388461,   0.        ,
                0.        ,  -0.07642193,   0.09473605,   0.12277104,
                3.84009611])
```



Through the following GBR try, we got r2_score of .807% on testing set:

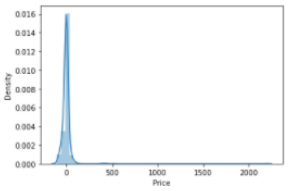
[170]:
```
#creating model instance with best params
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
x_train, x_test, y_train, y_test=train_test_split(x,Y,test_size=0.25,random_state=0)


model=GradientBoostingRegressor(alpha=0.09,
                                learning_rate=0.1,
                                max_depth=2,
                                min_samples_split=2,
                                min_samples_leaf=1,
                                n_estimators=10)

model.fit(x_train,y_train)
pred=model.predict(x_test)
test_score=r2_score(y_test,pred)
train_score=r2_score(y_train,model.predict(x_train))
if abs(train_score-test_score)<=0.3:
    print(model)
    print(abs(train_score-test_score))
    print("R2 score is: ", r2_score(y_test,pred))
    print("R2 score for train data: ", r2_score(y_train,model.predict(x_train)))
    print("Mean absolute error is: ", mean_absolute_error(y_test, pred))
    print("Mean squared error is: ", mean_squared_error(y_test, pred))
    print("Root mean squared error is: ", (mean_squared_error(y_test,pred)))
```

```
GradientBoostingRegressor(alpha=0.09, max_depth=2, n_estimators=10)
0.01883086798710487
R2 score is:  0.00807267331097794
R2 score for train data:  0.02690354129808281
Mean absolute error is:  19.93776027970236
Mean squared error is:  5609.210977128447
Root mean squared error is:  5609.210977128447
```

linearregre 1/68

[171]:
```
#Predicting through GBR again just to make sure we have the model till now:
prediction=model.predict(x_test)
sns.distplot(y_test-prediction)
```

[171]:
```
<AxesSubplot:xlabel='Price', ylabel='Density'>
```



[172]:
```
#Score result through GBR for TMIN
result_ontestfromtrainingset=r2_score(y_test,prediction)
abs(result_ontestfromtrainingset)
```

[172]:
```
0.00807267331097794
```

[174]:
```
#saving the model for Tmin
import pickle
filename='Finalmodel.pkl'
pickle.dump(model,open(filename,'wb'))
```

[178]:
```
result=r2_score(y_test,pred)
result #so, we see this r2_score is better than the last saved one:
```

[178]:
```
0.00807267331097794
```

[177]:
```
#creting dataframe sample for tmin on test data
Conclusion=pd.DataFrame([loaded_model.predict(x_test)[:],pred[:]],index=["Predicted","Original"])
Conclusion
```

[177]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Predicted	140.186726	131.16463	132.500287	140.186726	131.16463	131.465902	140.186726	140.186726	131.16463	140.186726	131.16463	140.186726	140.186726	140.186726	140.186726	132.500287	140.186726	132.500287	132.500287	140.186726	131.16463	1
Original	140.186726	131.16463	132.500287	140.186726	131.16463	131.465902	140.186726	140.186726	131.16463	140.186726	131.16463	140.186726	140.186726	140.186726	140.186726	132.500287	140.186726	132.500287	132.500287	140.186726	131.16463	1

Trying to handle the outliers and check the ML algorithms results again:

[112]:
```
outliers = []
def detect_outliers_iqr(data):
    data = sorted(data)
    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)
    # print(q1, q3)
    IQR = q3-q1
    lwr_bound = q1-(1.5*IQR)
    upr_bound = q3+(1.5*IQR)
    # print(lwr_bound, upr_bound)
    for i in data:
        if (i<lwr_bound or i>upr_bound):
            outliers.append(i)
    return outliers# Driver code
```

[114]:
```
sample_outliers = detect_outliers_iqr(df_new.Price)
print("Outliers from IQR method: ", sample_outliers)
```

```
Outliers from IQR method:  [29, 29, 29, 29, 34, 34, 37, 37, 41, 41, 41, 43, 44, 44, 44,
4, 54, 54, 54, 55, 55, 55, 55, 55, 56, 56, 56, 56, 56, 56, 57, 58, 58, 58, 58, 58, 59,
66, 67, 67, 67, 67, 67, 67, 67, 67, 67, 68, 68, 68, 69, 69, 69, 69, 69, 69, 69, 69,
3, 73, 73, 75, 75, 75, 75, 75, 76, 76, 78, 78, 78, 78, 78, 78, 78, 78, 78, 78, 78, 78,
81, 81, 81, 81, 81, 81, 81, 82, 82, 82, 82, 82, 82, 82, 82, 83, 83, 83, 83, 83, 83, 83,
4, 84, 84, 84, 84, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85,
87, 87, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 89, 89, 89, 89, 89, 89, 89, 89,
2, 92, 92, 92, 92, 92, 92, 92, 92, 92, 93, 93, 93, 93, 93, 94, 94, 94, 94, 94, 94, 94,
95, 95, 95, 96, 96, 96, 96, 96, 96, 96, 96, 96, 96, 96, 96, 97, 97, 97, 97, 97, 97, 97,
0, 100, 100, 100, 100, 100, 101, 101, 101, 101, 101, 101, 101, 101, 101, 101, 101, 101,
```

[123]:
```
df_new.Price
```

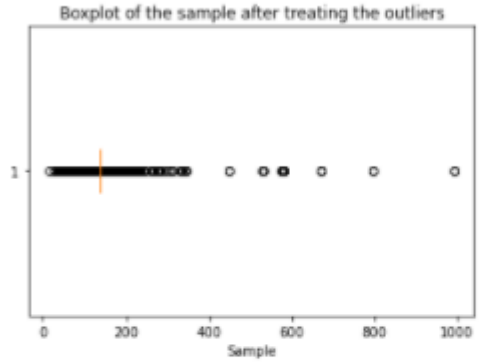
[123]:
```
0        29
1        34
2       136
3       136
4       136
       ...
1942    136
1943    136
1944    164
1945    166
1946    165
Name: Price, Length: 4701, dtype: int64
```

[140]:
```
median = np.median(df_new.Price)# Replace outliers with median
for i in sample_outliers:
    c = np.where(df_new.Price==i, 14, df_new.Price)
print("Sample: ", df_new.Price)
print("New array: ",c)
# print(x.dtype)
```

```
Sample:  0        29
1        34
2       136
3       136
4       136
       ...
1942    136
1943    136
1944    164
1945    166
1946    165
Name: Price, Length: 4701, dtype: int64
New array:  [ 29  34 136 ... 164 166 165]
```

[141]:
```
plt.boxplot(c, vert=False)
plt.title("Boxplot of the sample after treating the outliers")
plt.xlabel("Sample")
```

[141]:
```
Text(0.5, 0, 'Sample')
```



[145]:
```
df_new["Price"] = df_new["Price"].map(lambda i: np.log(i) if i > 0 else 0)
print(df_new['Price'].skew())
print(df_new['Price'].skew())
```

```
0.27180799346210205
0.27180799346210205
```

[146]:
```
#Replacing Outliers with Median Values

print(df_new['Price'].quantile(0.50))
print(df_new['Price'].quantile(0.95))
df_new['Price'] = np.where(df_new['Price'] > 325, 140, df_new['Price'])
df_new.describe()
```

```
4.912654885736052
5.2094861528414205
```

[146]:
```
#Replacing Outliers with Median Values

print(df_new['Price'].quantile(0.50))
print(df_new['Price'].quantile(0.95))
df_new['Price'] = np.where(df_new['Price'] > 325, 140, df_new['Price'])
df_new.describe()
```

```
4.912654885736052
5.2094861528414205
```

[146]:

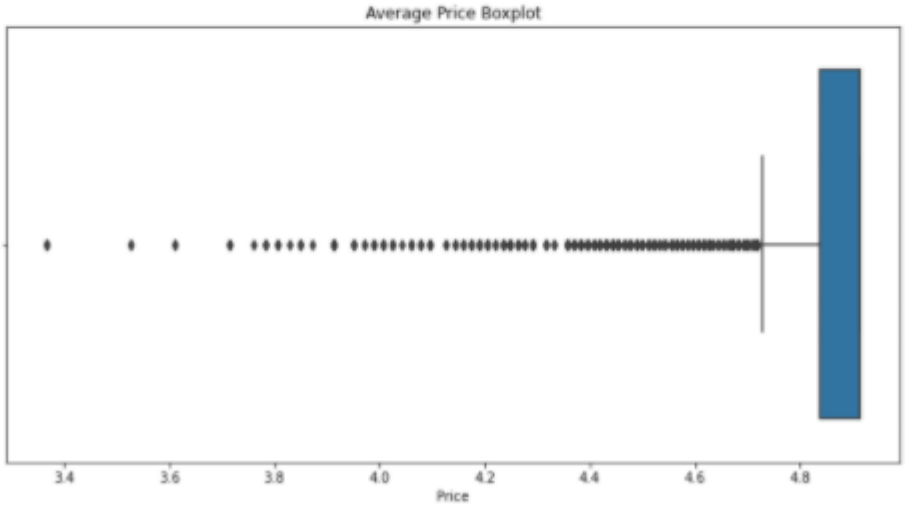
	Airline	Time dept	Direct	CitiesDept	Cities Arriv	Price	DeptDate	DeptMonth	DeptYear	ArrYear	Dep Time Mint	Arrv Time Hour	Arrv Time Mint	Travel hours
count	4701.000000	4701.000000	4701.000000	4701.000000	4701.000000	4701.000000	4701.0	4701.000000	4701.0	4701.0	4701.000000	4701.000000	4701.000000	4701.000000
mean	22.608807	16.843650	0.966390	1.154010	1.258456	4.881969	0.0	0.414167	0.0	0.0	5.202297	6.972984	6.003404	1.630504
std	11.236733	9.735547	0.180242	1.245289	1.346783	0.264615	0.0	0.492630	0.0	0.0	3.708891	4.571642	3.455306	1.501006
min	0.000000	0.000000	0.000000	0.000000	0.000000	3.367296	0.0	0.000000	0.0	0.0	0.000000	0.000000	0.000000	1.000000
25%	13.000000	7.000000	1.000000	0.000000	0.000000	4.912655	0.0	0.000000	0.0	0.0	1.000000	3.000000	3.000000	1.000000
50%	28.000000	19.000000	1.000000	0.000000	1.000000	4.912655	0.0	0.000000	0.0	0.0	5.000000	6.000000	7.000000	1.000000
75%	31.000000	25.000000	1.000000	2.000000	3.000000	4.912655	0.0	1.000000	0.0	0.0	9.000000	11.000000	9.000000	2.000000
max	40.000000	33.000000	1.000000	3.000000	3.000000	7.754910	0.0	1.000000	0.0	0.0	11.000000	14.000000	11.000000	8.000000

[148]:
```
df_new['Price'] #we can see several datapoints are replaced by median:
```

[148]:
```
0       3.367296
1       3.526361
2       4.912655
3       4.912655
4       4.912655
          ...
1942    4.912655
1943    4.912655
1944    5.099866
1945    5.111988
1946    5.105945
Name: Price, Length: 4701, dtype: float64
```

[151]:
```
#trying to drop outliers if left any:
s = df_new['Price']
iqr = (np.quantile(s, 0.75))-(np.quantile(s, 0.25))
upper_bound = np.quantile(s, 0.75)+(1.5*iqr)
lower_bound = np.quantile(s, 0.25)-(1.5*iqr)
df = df_new[(df_new['Price'] <= upper_bound)]
fig,ax = plt.subplots(figsize=(12,6))
fig = sns.boxplot(df.Price).set_title('Average Price Boxplot')
fig.figure.savefig('AP_drop.png')
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pass the followi
ing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



Also even using Isolation Forest:

[160]:
```
#Now, let's evaluate model performance with outliers removed using isolation forest:
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import IsolationForest
from sklearn.metrics import mean_absolute_error
'''# load the dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
df = read_csv(url, header=None)
# retrieve the array
data = df.values
# split into input and output elements
X, y = data[:, :-1], data[:, -1]'''
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(x, Y, test_size=0.33, random_state=1)
# summarize the shape of the training dataset
print(X_train.shape, y_train.shape)
# identify outliers in the training dataset
iso = IsolationForest(contamination=0.1)
yhat = iso.fit_predict(X_train)
# select all rows that are not outliers
mask = yhat != -1
#X_train, y_train = X_train[mask, :], y_train[mask]
# summarize the shape of the updated training dataset
print(X_train.shape, y_train.shape)
# fit the model
model = LinearRegression()
model.fit(X_train, y_train)
# evaluate the model
yhat = model.predict(X_test)
# evaluate predictions
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)
```

```
(3149, 13) (3149,)
(3149, 13) (3149,)
MAE: 22.302
```

[162]:
```
#Let's take radom state of 5

from sklearn.linear_model import LinearRegression
lr=LinearRegression()
X_train, X_test, Y_train, Y_test = train_test_split(x, Y, test_size=0.25, random_state=5)
lr.fit(X_train,Y_train)
```

[162]:
```
LinearRegression()
```

[163]:
```
#checking the R2 Score for the linear regression:
from sklearn.metrics import r2_score
pred_test=lr.predict(X_test)
print(r2_score(Y_test,pred_test))
```

```
0.03127120643709047
```

[164]:
```
# evaluate the model
yhatlr = lr.predict(X_test)
# evaluate predictions
maelr = mean_absolute_error(y_test[0:1176], yhatlr)
print('MAE: %.3f' % maelr)
```

```
MAE: 23.508
```

[166]:
```
#Regularization imports
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings('ignore')
```

[167]:
```
#we will cross validate with Lasso techniquue
from sklearn.linear_model import Lasso
parameters={ 'alpha':[.0001,.001,.01,.1,1,10], 'random_state': list(range(0,10))}
ls=Lasso()
clf=GridSearchCV(ls,parameters)
clf.fit(X_train,Y_train)
print(clf.best_params_)
```

```
{'alpha': 1, 'random_state': 0}
```

[168]:
```
#best cross fold is 2
cv_score=cross_val_score(clf,x,Y,cv=2)
cv_score
```

[168]:
```
array([-1.27658991, -0.02384537])
```

[169]:
```
cv_mean=cv_score.mean() #cv mean
cv_mean
```

[169]:
```
-0.6502176390549934
```

[170]:
```
#training with Lasso
ls=Lasso(alpha=1,random_state=0)
ls.fit(X_train,Y_train)
ls.score(X_train,Y_train)
pred_ls=ls.predict(X_test)
lss=r2_score(Y_test,pred_ls)
lss  #this is the BESTEST r2_score i could get in this case study. So LASSO is our final selected model.
```

[170]:
```
0.024420237647289067
```

[171]:
```
# grid search hyperparameters for the lasso algorithm
from sklearn.model_selection import RepeatedKFold
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
lasso_alphas = np.linspace(0, 0.2, 21)
lasso = Lasso()
grid = dict()
grid['alpha'] = lasso_alphas
gscv = GridSearchCV( \
    lasso, grid, scoring='neg_mean_absolute_error', \
    cv=cv, n_jobs=-1)
results = gscv.fit(X_train, Y_train)
print('MAE: %.5f' % results.best_score_)
print('Config: %s' % results.best_params_)  #this is the MAE i could get.
```

```
MAE: -20.71959
Config: {'alpha': 0.2}
```

[172]:
```
results.best_params_
```

[172]:
```
{'alpha': 0.2}
```

[173]:
```
#Lets calculate the score of the regularization Lasso:
ls=Lasso(alpha=0.2,random_state=0)
ls.fit(X_train,Y_train)
ls.score(X_train,Y_train)
pred_ls=ls.predict(X_test)
lss=r2_score(Y_test,pred_ls)
lss                              #the BESTEST r2 score till now. so we will select this model as it performs better than all others.
```

[173]:
```
0.030056939714088182
```

[102]:
```
#Lets calculate the score of the regularization Lasso:
from sklearn.linear_model import Lasso

ls=Lasso(alpha=0.2,random_state=0)
ls.fit(X_train,Y_train)
ls.score(X_train,Y_train)
pred_ls=ls.predict(X_test)
lss=r2_score(Y_test,pred_ls)
lss
```

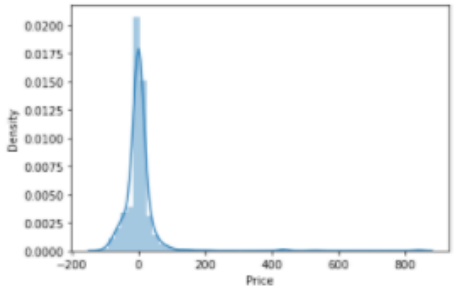
[102]:
```
0.030056939714088182
```

[175]:
```
result=ls.score(X_train,Y_train)
print(result)
```

```
0.018464882796334736
```

And the BESTEST result I got in this project was 3% on testing set as shown below:

```
[174]: result=ls.score(X_test,Y_test)
       print(result)   #the BESTEST r2 score  ON THE TEST DATA till now. so we will select this model as it performs better than all others.

       0.030056939714088182

[103]: # evaluate the model
       yhatls = ls.predict(X_test)
       # evaluate predictions
       maels = mean_absolute_error(Y_test, yhatls)
       print('MAE: %.3f' % maels)                #BETTER MAE COMPARED WITH OTHERS:

       MAE: 22.801

[104]: #In order to validate the model, let's see if the mean errors are less so the model is predicting with high performance
       from sklearn import metrics

       print('MAE:', metrics.mean_absolute_error(y_test[0:1176], yhatls))
       print('MSE:', metrics.mean_squared_error(y_test[0:1176], yhatls))
       print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test[0:1176], yhatls))) #LOWEST MEAN ERRORS:

       MAE: 23.26274597946242
       MSE: 3173.9016145319465
       RMSE: 56.33739090987394

[108]: #prediction on the test data and check residual:
       prediction=ls.predict(X_test)
       sns.distplot(Y_test-prediction)

[108]: <AxesSubplot:xlabel='Price', ylabel='Density'>
```




```
[112]: #r2 score of the predicted values of training values compared to actual price:
       result_ontestfromtrainingset=r2_score(Y_test,prediction)
       abs(result_ontestfromtrainingset)                    #BESTEST Result in this case:

[112]: 0.0068094429588089

[113]: #saving the model: #we will FINALLY SAVE this model as the final model because we have no more time to study and improve the algorithm results through other approaches:
       import pickle
       filename='Flight.pkl'
       pickle.dump(ls,open(filename,'wb'))

[114]: #loading the model and checking the accuracy on the test data:
       import pickle
       loaded_model=pickle.load(open('Flight.pkl','rb'))
       result=loaded_model.score(X_test,Y_test)
       print(result)

       0.030056939714088182

[118]: #3% is a HIGHEST accuracy I got on the test data. Not good, but have no more time to study more and different approaches:
       Conclusion=pd.DataFrame([loaded_model.predict(X_test)[:],Y_test[:]],index=["Predicted","Original"])
       Conclusion
```

[118]:

	0	1	2	3	4	5	6	7	8	9	...	1166	1167	1168	1169	1170	1171	1172	1173	1174	1175
Predicted	140.461736	147.052833	128.905917	128.905807	131.921172	139.023154	142.629705	128.380433	143.481882	139.023223	...	126.737624	128.380316	139.02312	125.553836	128.899176	127.297075	139.023154	139.277879	140.396797	125.55376
Original	136.000000	78.000000	136.000000	153.000000	136.000000	159.000000	135.000000	152.000000	190.000000	136.000000	...	136.000000	136.000000	181.00000	136.000000	136.000000	136.000000	136.000000	136.000000	205.000000	174.00000

2 rows × 1176 columns

Model scored 3% which was the maximum we in this project and FINALLY we decide to FINAL SAVE THIS Model as it has the maximum score of whole project.

Even though, if we try to manage the outliers through caping as shown in the following screenshot, the results won't change significantly:

ok_fin _ max i got as accuracy is 3%...need to learn more on handling the outliers which are afecting the resukts and accuracy of the results.

One last try i would have done is to drop outliers from only the columns which includes outliers. But have no more time as the deadline is tomorrow. So will finish here and start writing the documentation.

ok, need to stop here! evengthough i could have got better accuracy when dropping the outliers only from the columns including outliers.ok, need to stop here! no more time!

[130]:
```
x
```

[130]:

	Airline	Time_dept	Direct	CitiesDept	Cities_Arriv	DeptDate	DeptMonth	DeptYear	ArrYear	Dep_Time_Mint	Arrv_Time_Hour	Arrv_Time_Mint	Travel_hours
0	32	25	1	0	3	0	0	0	0	0	13	10	1
1	32	29	1	1	0	0	0	0	0	0	0	9	1
2	28	7	1	0	3	0	0	0	0	10	4	8	1
3	28	29	1	1	0	0	0	0	0	0	0	9	1
4	13	25	1	0	3	0	0	0	0	0	13	10	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1942	5	27	1	0	2	0	1	0	0	11	14	11	2
1943	6	25	1	2	0	0	1	0	0	0	13	9	1
1944	31	30	1	0	2	0	1	0	0	4	1	6	2
1945	6	4	1	3	0	0	1	0	0	5	3	1	1
1946	2	11	1	0	2	0	1	0	0	0	6	0	2

4701 rows × 13 columns

[131]:
```
print("Highest allowed",df_new['Airline'].mean() + 3*df_new['Airline'].std())
print("Lowest allowed",df_new['Airline'].mean() - 3*df_new['Airline'].std())
```

```
Highest allowed 56.31900573324833
Lowest allowed -11.101392459476795
```

[132]:
```
df_new[(df_new['Airline'] > 56.31) | (df_new['Airline'] < -11.1)]
```

[132]:

Airline	Time_dept	Direct	CitiesDept	Cities_Arriv	Price	DeptDate	DeptMonth	DeptYear	ArrYear	Dep_Time_Mint	Arrv_Time_Hour	Arrv_Time_Mint	Travel_hours

[133]:
```
print("Highest allowed",df_new['Time_dept'].mean() + 3*df_new['Time_dept'].std())
print("Lowest allowed",df_new['Time_dept'].mean() - 3*df_new['Time_dept'].std())
```

[133]:

```
Highest allowed 46.05029025886594
Lowest allowed -12.362989684520063
```

[134]:
```
df_new[(df_new['Time_dept'] > 46.050) | (df_new['Time_dept'] < -12.36)]
```

[134]:

Airline	Time_dept	Direct	CitiesDept	Cities_Arriv	Price	DeptDate	DeptMonth	DeptYear	ArrYear	Dep_Time_Mint	Arrv_Time_Hour	Arrv_Time_Mint	Travel_hours

[135]:
```
print("Highest allowed",df_new['CitiesDept'].mean() + 3*df_new['CitiesDept'].std())
print("Lowest allowed",df_new['CitiesDept'].mean() - 3*df_new['CitiesDept'].std())
```

```
Highest allowed 4.8898759793921585
Lowest allowed -2.581856409087967
```

[136]:
```
df_new[(df_new['CitiesDept'] > 4.889) | (df_new['CitiesDept'] < -2.581)]
```

[136]:

Airline	Time_dept	Direct	CitiesDept	Cities_Arriv	Price	DeptDate	DeptMonth	DeptYear	ArrYear	Dep_Time_Mint	Arrv_Time_Hour	Arrv_Time_Mint	Travel_hours

[137]:
```
print("Highest allowed",df_new['Direct'].mean() + 3*df_new['Direct'].std())
print("Lowest allowed",df_new['Direct'].mean() - 3*df_new['Direct'].std())
```

```
Highest allowed 1.5071159549081874
Lowest allowed 0.42566430461106375
```

[137]:
```
print("Highest allowed",df_new['Cities_Arriv'].mean() + 3*df_new['Cities_Arriv'].std())
print("Lowest allowed",df_new['Cities_Arriv'].mean() - 3*df_new['Cities_Arriv'].std())
```

```
Highest allowed 5.298804997808499
Lowest allowed -2.78189370233945
```

[139]:
```
df_new[(df_new['Cities_Arriv'] > 5.2988) | (df_new['Cities_Arriv'] < -2.7818)]
```

[139]:

Airline	Time_dept	Direct	CitiesDept	Cities_Arriv	Price	DeptDate	DeptMonth	DeptYear	ArrYear	Dep_Time_Mint	Arrv_Time_Hour	Arrv_Time_Mint	Travel_hours

```
[139]: df_new[(df_new['Cities_Arriv'] > 5.2988) | (df_new['Cities_Arriv'] < -2.7818)]
```

[139]:

	Airline	Time_dept	Direct	CitiesDept	Cities_Arriv	Price	DeptDate	DeptMonth	DeptYear	ArrYear	Dep_Time_Mint	Arrv_Time_Hour	Arrv_Time_Mint	Travel_hours

```
[140]: print("Highest allowed",df_new['DeptDate'].mean() + 3*df_new['DeptDate'].std())
       print("Lowest allowed",df_new['DeptDate'].mean() - 3*df_new['DeptDate'].std())
```

```
Highest allowed 0.0
Lowest allowed 0.0
```

```
[141]: df_new[(df_new['DeptDate'] > 0) | (df_new['DeptDate'] < 0)]
```

[141]:

	Airline	Time_dept	Direct	CitiesDept	Cities_Arriv	Price	DeptDate	DeptMonth	DeptYear	ArrYear	Dep_Time_Mint	Arrv_Time_Hour	Arrv_Time_Mint	Travel_hours

```
[142]: print("Highest allowed",df_new['DeptMonth'].mean() + 3*df_new['DeptMonth'].std())
       print("Lowest allowed",df_new['DeptMonth'].mean() - 3*df_new['DeptMonth'].std())
```

```
Highest allowed 1.8920573119401312
Lowest allowed -1.0637229150033092
```

```
[143]: df_new[(df_new['DeptMonth'] > 1.8920573) | (df_new['DeptMonth'] < -1.063)]
```

[143]:

	Airline	Time_dept	Direct	CitiesDept	Cities_Arriv	Price	DeptDate	DeptMonth	DeptYear	ArrYear	Dep_Time_Mint	Arrv_Time_Hour	Arrv_Time_Mint	Travel_hours

```
[144]: print("Highest allowed",df_new['Dep_Time_Mint'].mean() + 3*df_new['Dep_Time_Mint'].std())
       print("Lowest allowed",df_new['Dep_Time_Mint'].mean() - 3*df_new['Dep_Time_Mint'].std())
```

```
Highest allowed 16.32897164621282
Lowest allowed -5.924376879141985
```

```
[145]: df_new[(df_new['Dep_Time_Mint'] > 16.328) | (df_new['Dep_Time_Mint'] < -5.92437)]
```

[145]:

	Airline	Time_dept	Direct	CitiesDept	Cities_Arriv	Price	DeptDate	DeptMonth	DeptYear	ArrYear	Dep_Time_Mint	Arrv_Time_Hour	Arrv_Time_Mint	Travel_hours

```
[154]: #Now let's split the drop Price as target, and 'DeptDate','DeptYear','ArrYear' as these have unique 1 value:
       x=df_new.drop(['Price','DeptDate','DeptYear','ArrYear'],axis=1)
       Y=df_new['Price']
```

```
[155]: #Modeling-- OKRR
       #We now proceed to the main step of our machine learning, fitting the model and predicting the outputs. We fit the data into multiple regression

       #Importing libraries

       from sklearn.linear_model import LinearRegression
       from sklearn.tree import DecisionTreeRegressor
       from sklearn.neighbors import KNeighborsRegressor
       from sklearn.svm import SVR
       from sklearn.model_selection import train_test_split
       from sklearn.metrics import r2_score, mean_absolute_error,mean_squared_error
```

```
[156]: # evaluate model performance with outliers removed using isolation forest
       from pandas import read_csv
       from sklearn.model_selection import train_test_split
       from sklearn.linear_model import LinearRegression
       from sklearn.ensemble import IsolationForest
       from sklearn.metrics import mean_absolute_error
       '''# load the dataset
       url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
       df = read_csv(url, header=None)
       # retrieve the array
       data = df.values
       # split into input and output elements
       X, y = data[:, :-1], data[:, -1]'''
       # split into train and test sets
       X_train, X_test, y_train, y_test = train_test_split(x, Y, test_size=0.33, random_state=1)
       # summarize the shape of the training dataset
       print(X_train.shape, y_train.shape)
       # identify outliers in the training dataset
       iso = IsolationForest(contamination=0.1)
       yhat = iso.fit_predict(X_train)
       # select all rows that are not outliers
       mask = yhat != -1
       #X_train, y_train = X_train[mask, :], y_train[mask]
       # summarize the shape of the updated training dataset
       print(X_train.shape, y_train.shape)
       # fit the model
       model = LinearRegression()
       model.fit(X_train, y_train)
       # evaluate the model
```

```
# evaluate the model
yhat = model.predict(X_test)
# evaluate predictions
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)
```

```
(3149, 10) (3149,)
(3149, 10) (3149,)
MAE: 22.302
```

[157]:
```
x_train=X_train
y_train=y_train
x_test=X_test
model.fit(x_train,y_train)
pred=model.predict(x_test)
test_score=r2_score(y_test,pred)
train_score=r2_score(y_train,model.predict(x_train))
if abs(train_score-test_score)<=0.3:
    print(model)
    print(abs(train_score-test_score))
    print("R2 score is: ", r2_score(y_test,pred))
    print("R2 score for train data: ", r2_score(y_train,model.predict(x_train)))
    print("Mean absolute error is: ", mean_absolute_error(y_test, pred))
    print("Mean squared error is: ", mean_squared_error(y_test, pred))
    print("Root mean squared error is: ", (mean_squared_error(y_test,pred)))
```

```
LinearRegression()
0.005578364089640742
R2 score is:  0.02351419916815012
R2 score for train data:  0.017935835078509377
Mean absolute error is:  22.302395673754685
Mean squared error is:  3000.4058698973145
Root mean squared error is:  3000.4058698973145
```

[158]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
#X_train, X_test, Y_train, Y_test = train_test_split(x, Y, test_size=0.25, random_state=5)
lr.fit(X_train,y_train)
```

[158]:
```
LinearRegression()
```

[166]:
```
#Cross validation of the model Linear regression:
yhat_test = lr.predict(X_test)
yhat_train = lr.predict(X_train)
Train_accuracy=r2_score(y_train,yhat_train)
Test_accuracy=r2_score(y_test,yhat_test)

from sklearn.model_selection import cross_val_score
for j in range(2,20):
    cv_score=cross_val_score(lr,x,Y,cv=j)
    cv_mean=cv_score.mean()
    print(f"At cross fold {j} the CV score is {cv_mean} and accuracy score for training is {Train_accuracy} and accuracy for the testing is {Test_accuracy}")
    print("\n")
```

highest allowed

```
At cross fold 2 the CV score is -7.185026157400266 and accuracy score for training is 0.017935835078509377 and accuracy for the testing is 0.02351419916815012


At cross fold 3 the CV score is -1.252553232701415 and accuracy score for training is 0.017935835078509377 and accuracy for the testing is 0.02351419916815012


At cross fold 4 the CV score is -0.6933354188149967 and accuracy score for training is 0.017935835078509377 and accuracy for the testing is 0.02351419916815012


At cross fold 5 the CV score is -0.7924847822796303 and accuracy score for training is 0.017935835078509377 and accuracy for the testing is 0.02351419916815012


At cross fold 6 the CV score is -0.8047220113877119 and accuracy score for training is 0.017935835078509377 and accuracy for the testing is 0.02351419916815012


At cross fold 7 the CV score is -0.8213966710901703 and accuracy score for training is 0.017935835078509377 and accuracy for the testing is 0.02351419916815012


At cross fold 8 the CV score is -0.7425248395881263 and accuracy score for training is 0.017935835078509377 and accuracy for the testing is 0.02351419916815012


At cross fold 9 the CV score is -0.8816816321431924 and accuracy score for training is 0.017935835078509377 and accuracy for the testing is 0.02351419916815012


At cross fold 10 the CV score is -0.8009029200354034 and accuracy score for training is 0.017935835078509377 and accuracy for the testing is 0.02351419916815012


At cross fold 11 the CV score is -0.7569631715627768 and accuracy score for training is 0.017935835078509377 and accuracy for the testing is 0.02351419916815012
```

[164]:
```
# evaluate the model thorugh mean absolute error
yhatlr = lr.predict(X_test)
# evaluate predictions
maelr = mean_absolute_error(y_test, yhatlr)
print('MAE: %.3f' % maelr)
```

```
MAE: 22.302
```

[165]:
```
print(lr)
print("R2 score is: ", r2_score(y_test,yhatlr))
print("R2 score for train data: ", r2_score(y_train,lr.predict(x_train)))
print("Mean absolute error is: ", mean_absolute_error(y_test, yhatlr))
print("Mean squared error is: ", mean_squared_error(y_test, yhatlr))
print("Root mean squared error is: ", (mean_squared_error(y_test,yhatlr)))
```

```
LinearRegression()
R2 score is:  0.02351419916815012
R2 score for train data:  0.017935835078509377
Mean absolute error is:  22.302395673754685
Mean squared error is:  3000.4058698973145
Root mean squared error is:  3000.4058698973145
```

[169]:
```
#we will cross validate with Lasso techniquue
from sklearn.linear_model import Lasso
parameters={ 'alpha':[.0001,.001,.01,.1,1,10], 'random_state': list(range(0,10))}
ls=Lasso()
clf=GridSearchCV(ls,parameters)
clf.fit(X_train,y_train)
print(clf.best_params_)
```

```
{'alpha': 1, 'random_state': 0}
```

[172]:
```
#checking the r2 score with Lasso technique:
ls=Lasso(alpha=1,random_state=0)
ls.fit(X_train,y_train)
ls.score(X_train,y_train)
pred_ls=ls.predict(X_test)
lss=r2_score(y_test,pred_ls)
lss
```

[172]:
```
0.017532126090745503
```

Even, after caping the outliers, we haven't improved our BESTEST score.

- Key Metrics for success in solving problem under consideration

  Basically, I used the 3 errors metrics and R2 score alongside the test score in order to compare and choose the best algorithms as shown below:

[96]:
```
#Modeling-- OKRR
#We now proceed to the main step of our machine learning, fitting the model and predicting the output

#Importing libraries

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error,mean_squared_error
```

[97]:
```
# evaluate model performance with outliers removed using isolation forest
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import IsolationForest
from sklearn.metrics import mean_absolute_error
'''# load the dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
df = read_csv(url, header=None)
# retrieve the array
data = df.values
# split into input and output elements
X, y = data[:, :-1], data[:, -1]'''
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(x, Y, test_size=0.33, random_state=1)
# summarize the shape of the training dataset
print(X_train.shape, y_train.shape)
# identify outliers in the training dataset
iso = IsolationForest(contamination=0.1)
yhat = iso.fit_predict(X_train)
# select all rows that are not outliers
mask = yhat != -1
#X_train, y_train = X_train[mask, :], y_train[mask]
# summarize the shape of the updated training dataset
print(X_train.shape, y_train.shape)
# fit the model
model = LinearRegression()
model.fit(X_train, y_train)
# evaluate the model
yhat = model.predict(X_test)
# evaluate predictions
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)
```

```
(3149, 13) (3149,)
(3149, 13) (3149,)
MAE: 22.302
```

For results comparison, we calculated R2 score, MAE and RMSE as the error evaluation metrics as follows:

[101]:
```
x_train=X_train
y_train=y_train
x_test=X_test
model.fit(x_train,y_train)
pred=model.predict(x_test)
test_score=r2_score(y_test,pred)
train_score=r2_score(y_train,model.predict(x_train))
if abs(train_score-test_score)<=0.3:
    print(model)
    print(abs(train_score-test_score))
    print("R2 score is: ", r2_score(y_test,pred))
    print("R2 score for train data: ", r2_score(y_train,model.predict(x_train)))
    print("Mean absolute error is: ", mean_absolute_error(y_test, pred))
    print("Mean squared error is: ", mean_squared_error(y_test, pred))
    print("Root mean squared error is: ", (mean_squared_error(y_test,pred)))
```

```
LinearRegression()
0.005578364089641075
R2 score is:  0.02351419916815012
R2 score for train data:  0.017935835078509044
Mean absolute error is:  22.302395673754685
Mean squared error is:  3000.4058698973145
Root mean squared error is:  3000.4058698973145
```

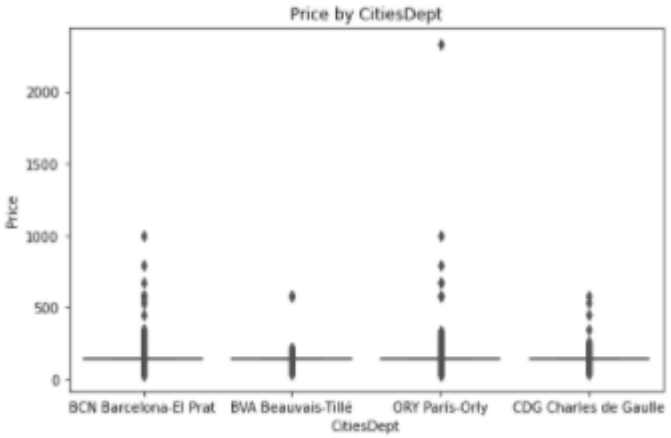
- Visualizations

  Mention all the plots made along with their pictures and what were the inferences and observations obtained from those. Describe them in detail.

  If different platforms were used, mention that as well.

```
[24]: #boxplotting the price by departure city:
      plt.figure(figsize=(8,5))
      sns.boxplot(x='CitiesDept',y='Price',data=df, palette='rainbow')
      plt.title("Price by CitiesDept")
```

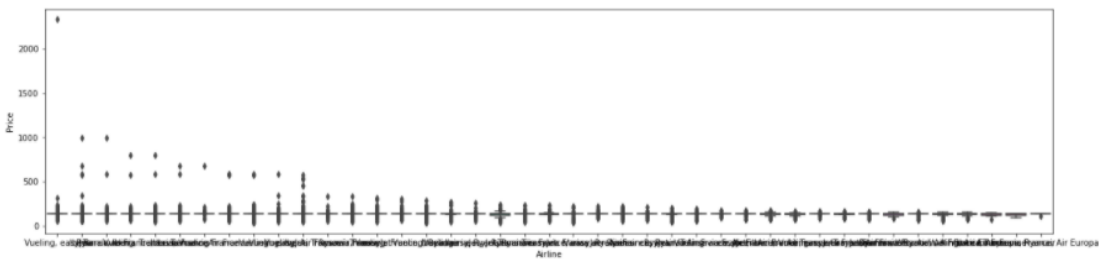
```
[24]: Text(0.5, 1.0, 'Price by CitiesDept')
```



We see that flights departing from Barcelona and Orly airport sometimes have a bit higher price than the other 2 airport. We also see some outliers in caseof departing from Orly Paris Airport as we can see a data dot which is located for price more than 200 euros and that is way far from the median prices of the airport PRLY Paris and it's the only outlier we have in Price when comparing the prices by the departure cities.

```
[25]: #boxplotting the Prices by Airlines:
      plt.figure(figsize=(22,5))
      sns.boxplot(x='Airline', y='Price', data=df.sort_values('Price',ascending=False))
```

```
[25]: <AxesSubplot:xlabel='Airline', ylabel='Price'>
```

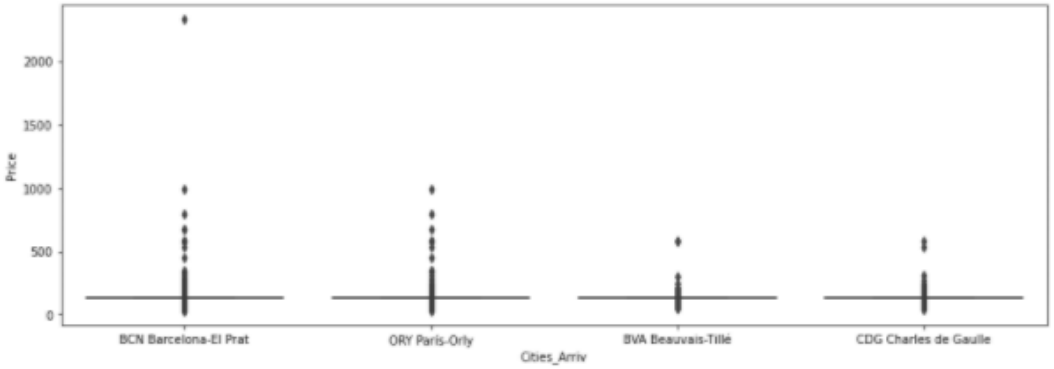


We can see all airines have similar price median but some airlines have few cases of higher flight prices compared to others.

We can come up with a conclusion all airlines had almost similar median with minimal fluctuations.

```
[27]: #boxplotting the flight price by Arrival city:
      plt.figure(figsize=(15,5))
      sns.boxplot(x='Cities_Arriv',y='Price',data=df.sort_values('Price',ascending=False))
```

```
[27]: <AxesSubplot:xlabel='Cities_Arriv', ylabel='Price'>
```

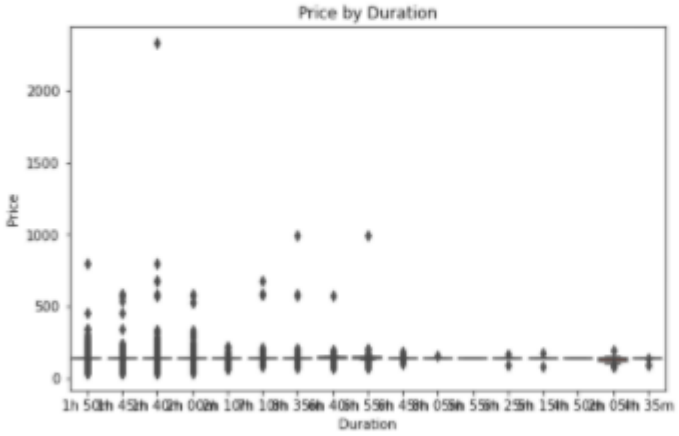


We see that flights arriving to Barcelona and Orly airport sometimes have a bit higher price than the other 2 airport. We also see some outliers in caseof arriving to Barcelona Airport as we can see a data dot which is located for price more than 200 euros and that is way far from the median prices of the airport Barcelona and it's the only outlier we have in Price when comparing the prices by the arrival cities.

It must be that same case of flight that departs from ORLY and get landed in Barcelona as there is just one outlier instance.

```
[28]: #boxplotting the flight price by Duration:
plt.figure(figsize=(8,5))
sns.boxplot(x='Duration',y='Price',data=df, palette='rainbow')
plt.title("Price by Duration")
```

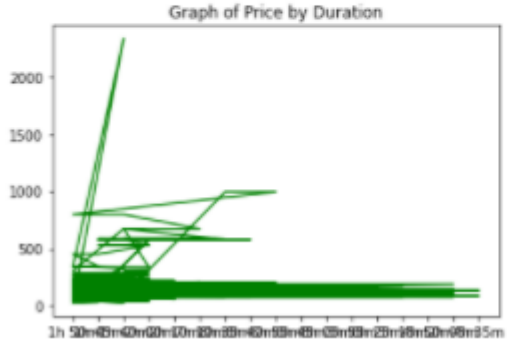
```
[28]: Text(0.5, 1.0, 'Price by Duration')
```



This graph is not that much useful to analize. Let's try another way to visualze the same data.

```
[29]: #plotting the flight price by Duration:
fig = plt.figure()
ax = plt.axes()

x = df['Duration']
y=df['Price']
plt.title('Graph of Price by Duration')
ax.plot(x, y, 'g', label='Price')
```

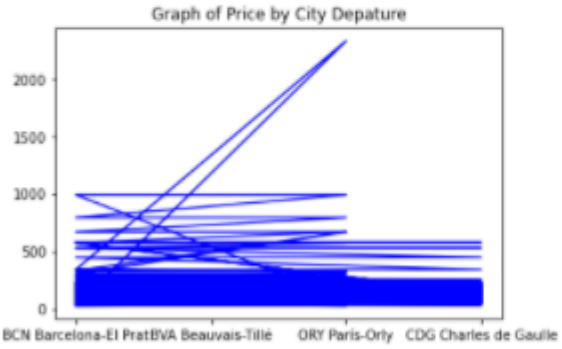
```
[29]: [<matplotlib.lines.Line2D at 0x7efe3cda2a90>]
```



No difference in showing better the prices by duration. But let's see the previous 2 visualizations and try to describe what's happening here: Here we can see the price gets mostly betwenn 0 and 400. The median is similar in all cases independently of the Duration.

[30]:
```
#Now let's plot the Price variation by Departure City :
fig = plt.figure()
ax = plt.axes()

x = df['CitiesDept']
y=df['Price']
plt.title('Graph of Price by City Depature')
ax.plot(x, y, 'b', label='Price')
```

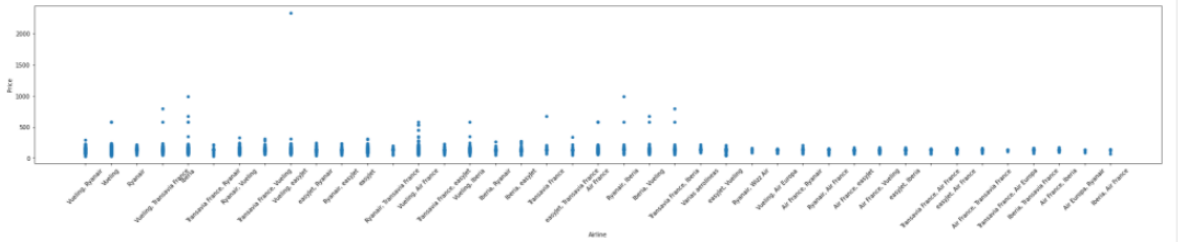
[30]: [<matplotlib.lines.Line2D at 0x7efe3cce0c50>]



Here we do not see this big much difference in price by departure city as long as we consider our previous comments.

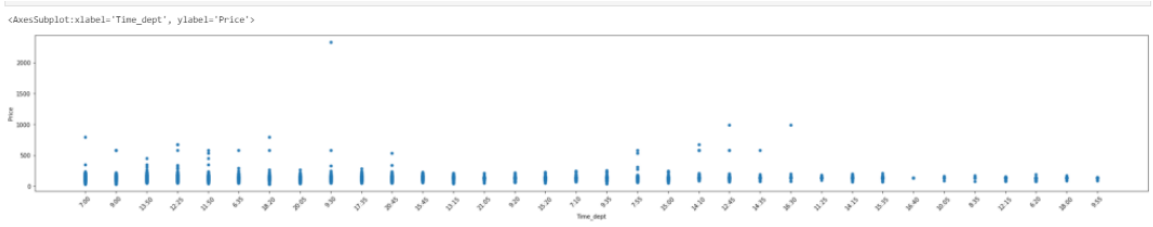
[34]:
```
#scatterplotting Price by Airlines:
#sns.scatterplot(data=df, x=col, y="Price")
df.plot(kind ="scatter", x= "Airline", y = "Price", rot=45, figsize=(35,5))
```

[34]: <AxesSubplot:xlabel='Airline', ylabel='Price'>

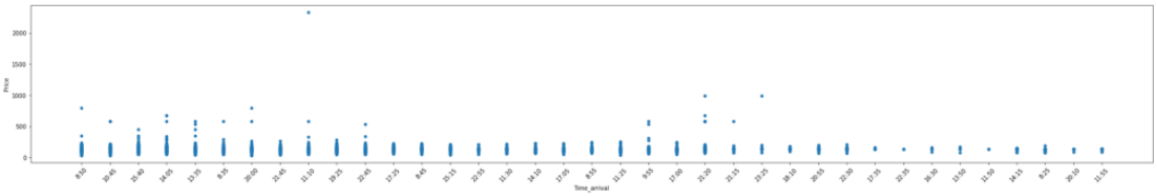


We see almost all airlines have similar median flights prices except few ones which sometimes have higher prices. These exceptions occurr with airlines like flights that are combined such like Vueling, Transavia France Iberia and also with Vueling and Ryanair or even when taking flights with Ryanair and Transvia France. We also have few higher prices when combining airlines like Easyjet, Transavia France and Air France. Also few higher flights price when taking flights with Ryanair and Iberia or even Iberia and Vueling.

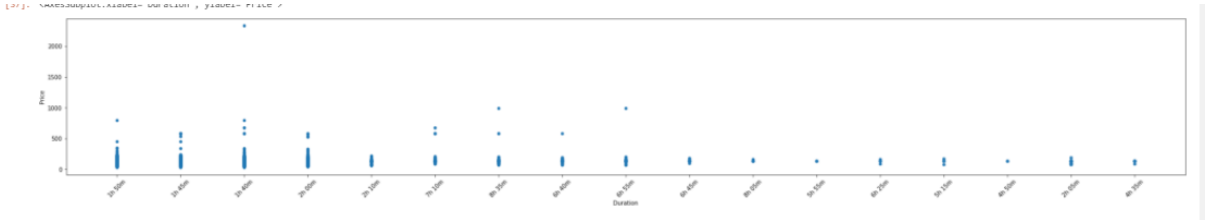
[35]: <AxesSubplot:xlabel='Time_dept', ylabel='Price'>



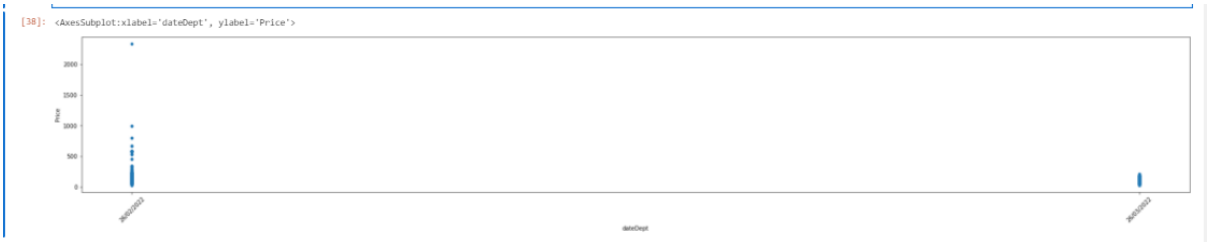
Regarding the departure time, we have few timings that may have higher prices than the mean and which are flights departing at 7am, 9am, 13:50pm, 12:25 pm, 11:50am, 18:20, 9:30am, 20:45, 7:55am, 14:10, 12:45pm, 14:35 and 16:30h.



Similar situation happens with Arrival Time, we have Price around the mean and the median but we have also Arrival Time where we have a bit higher prices such as in timings like 8:50, 10:45 (one case), 15:40, 14:05, 13:35,8:35 (one case), 20h (2 cases), 21:45, 11:10, 22:45 (2 cases), 9:55, 21:20, 21:15 (one ase) and 23:25 (one case).

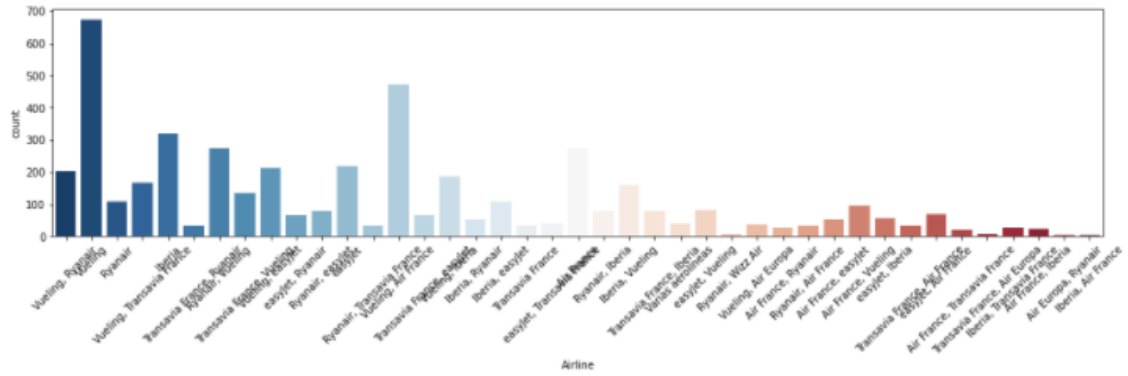


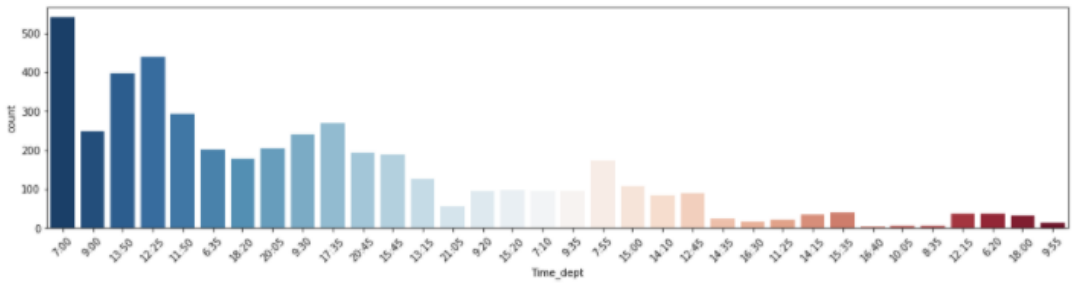
When comparing Duration, it can be studied and analyzed the same as tge departure and arrival time. Almost al the prices are between 0 and 500 euros except some cases where we have a bit higher prices such as duration of 1h 50min, 1h 45m, 1h 40m (we have 1 outlier here), 2h, 7h 10m,, 8h 35m, 6h 40m and 6h 55m.

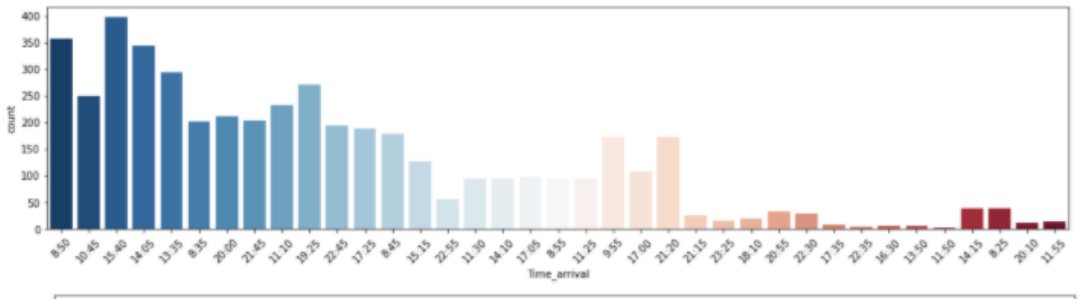


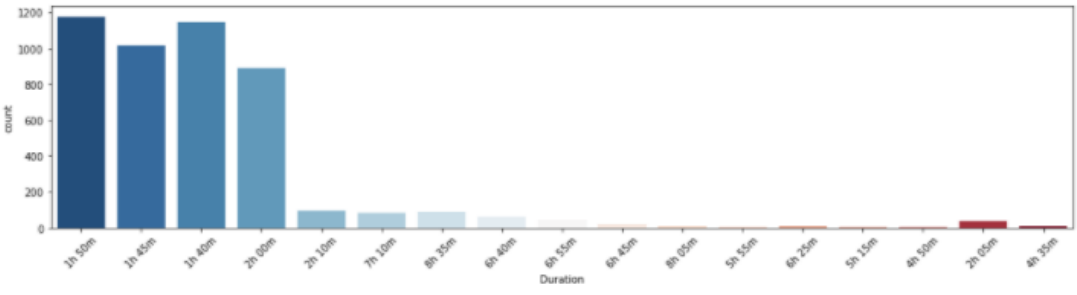
As we said, we have dataset including flights of Saturdays 26th Feb and 26th March.

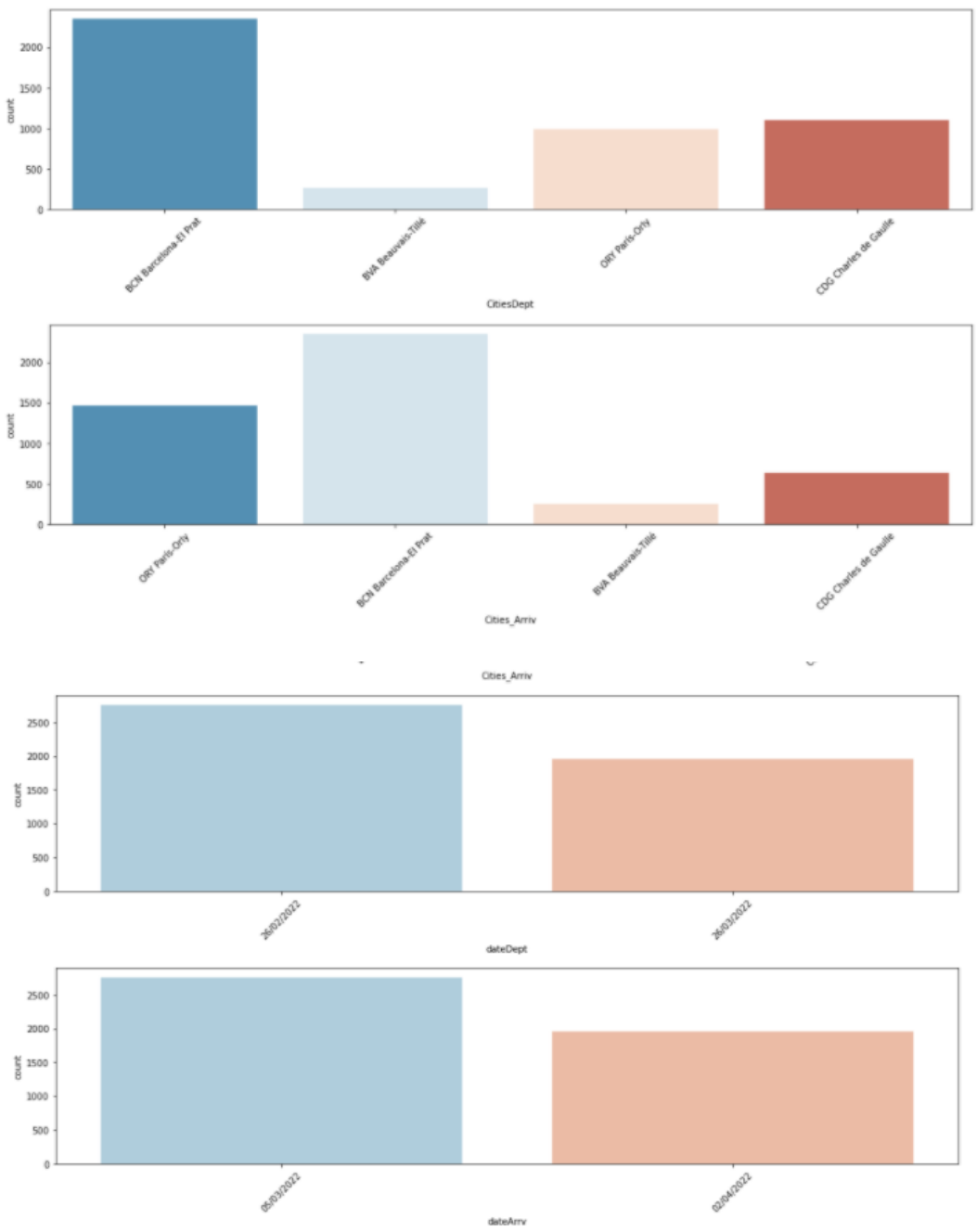
[36]:
```
#Let's plot the counting of each unique category in each column:
for col in colsexceptPrice:
    plt.figure(figsize=(18,4))
    plt.xticks(rotation=45)
    sns.countplot(x=col, data=df,palette='RdBu_r')
```











Here we can see easily, Vueling alone has the highest number of flights. Then in second position we have combo of Ryanair and Transavia France which comes in 2nd place and get the most flights after Vueling.

Then, in third place, we have combo of Vueling and Transavia France which gets most flights in third place.

When checking the flights counts by Time of departure, we see 7am has the highest number of seats available. Then, after that, in second place

comes flight that departs at 12:25h. And in 3rd place, comes the flight that departs at 13:50.

When comparing the lowest number of seats available, we see that we have 16:40,10:05,8:35h.

When checking the flights counts by Arrival Time, we see 15:40h has the highest number of seats available. Then, after that, in second place comes flight that arrives at 08:50h. And in 3rd place, comes the flight that departs at 14:05.

When comparing the lowest number of seats available, we see that we have 22:35,16:30,13:50 and 11:50.

As these flights are between Barcelona and Paris, it is obvious that most of the flights would be direct flights. There is just one exception which has 1 stay in between.

When checking flights by duration, we can see flights with duration etween 1:45h and 2h are the most used and available in the website.

When checking flights by Departure Cities, we see Barcelona gets the 1st position as the city from which the flights gets mostly departed as Barcelona is the main Airport in Spain. And in sencond place, from France we have 3 options as Departure Airports but the french airport which gets the position when comparing the flight departure is Charles de Gaulle which is the main airport of Paris.

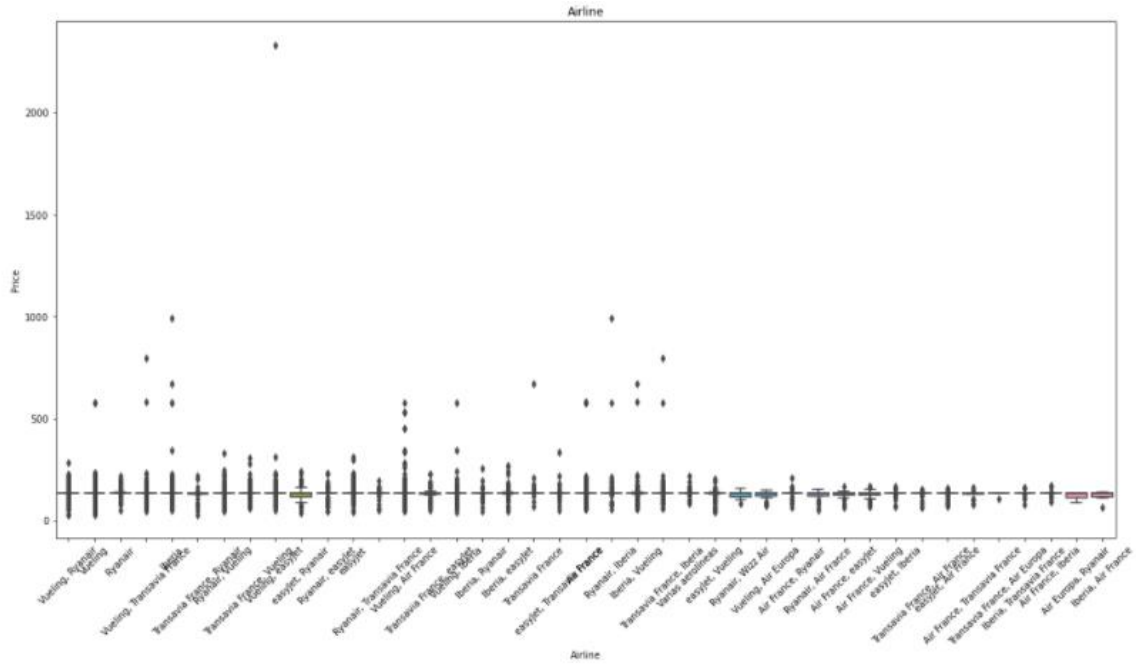
When checking flights by Arrival Cities, again we see Barcelona gets the 1st position as the city to which flights mostly arrives as Barcelona is the main Airport in Spain. And in sencond place, from France we have 3 options as Departure Airports but the french airport which gets the position when comparing the flight departure is Paris-Orly which is the second main airport of Paris.

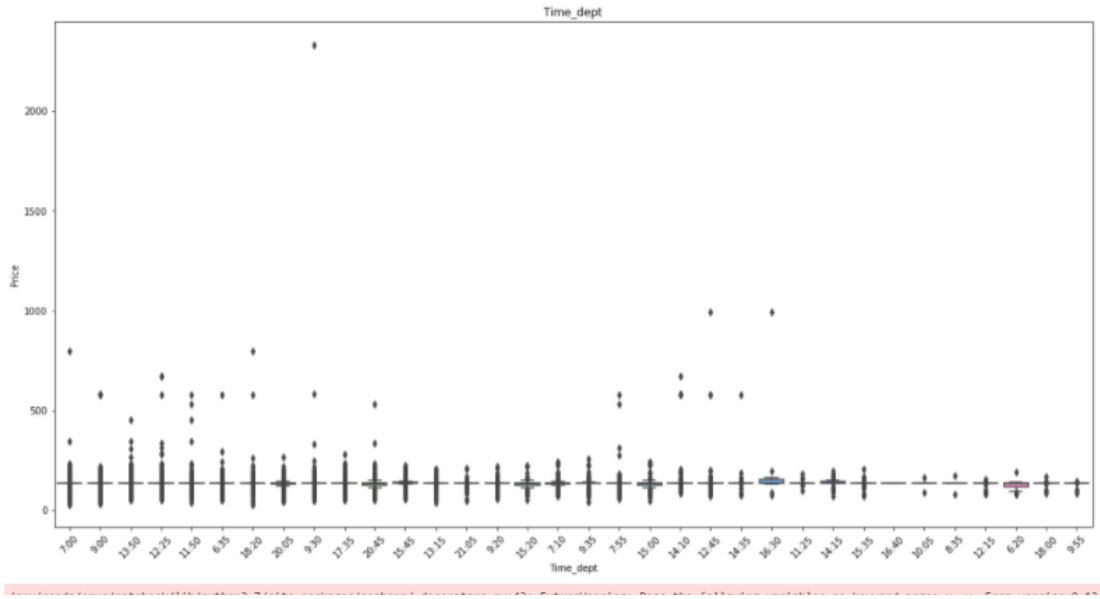
Regarding the dates of depart and arrival, as we said, we have only two dates of departure and two dates for arrival so that we could compare flights that are close to be flown and also flights far to be taken.

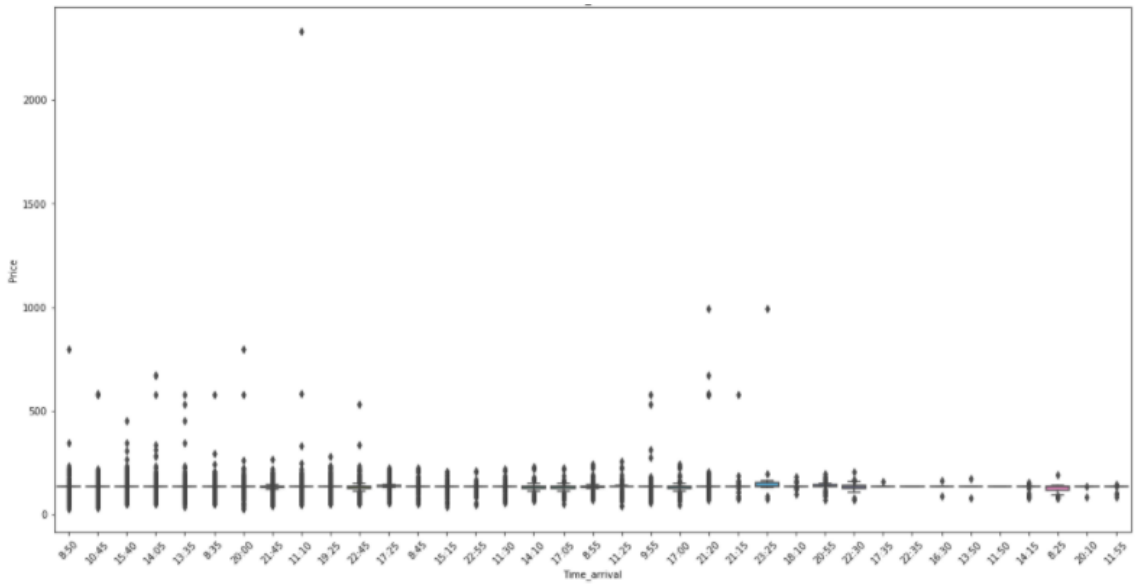
For flights price comparison, we took February depature date 26/02 and back on March, 05th

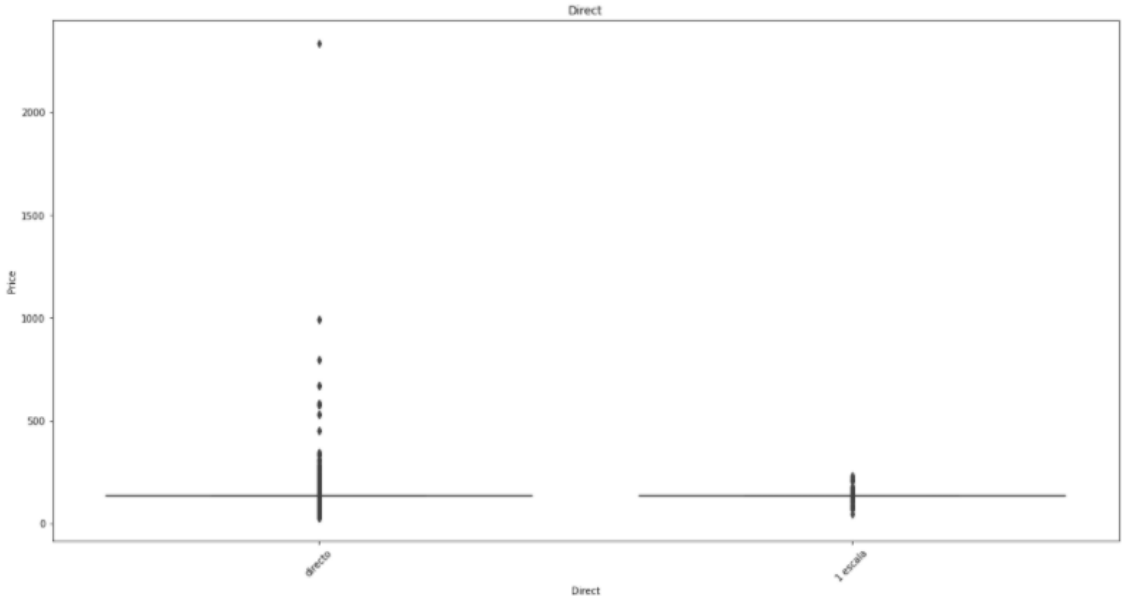
For flights a bit in far in future, we took 26/03 and bak on April, 2nd.

Both departure days are Saturdays so that we can compare the same day of week how the price varies.

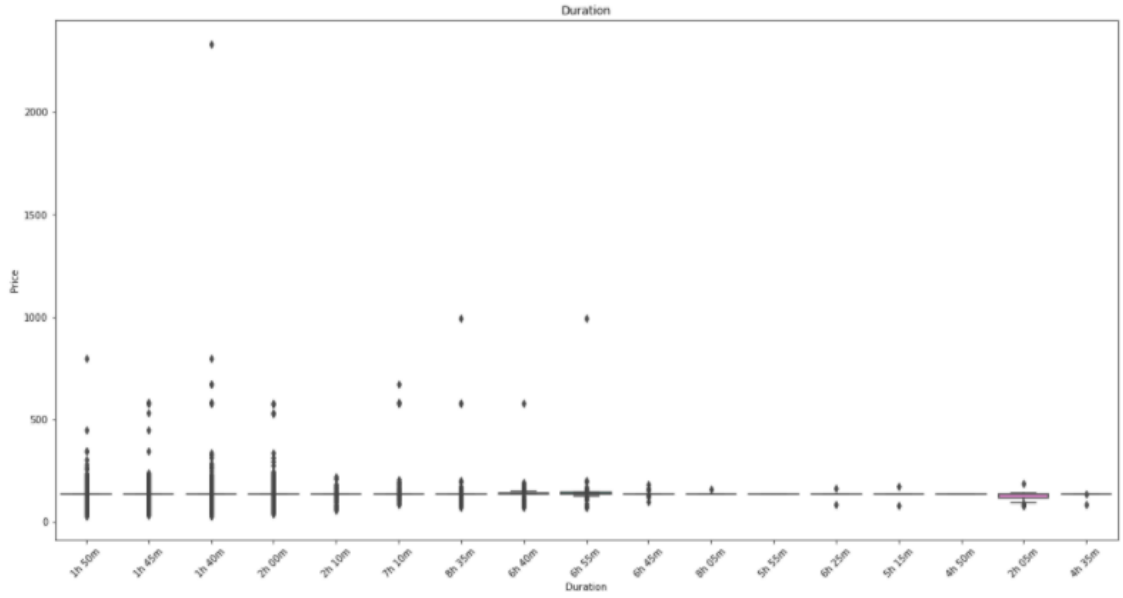






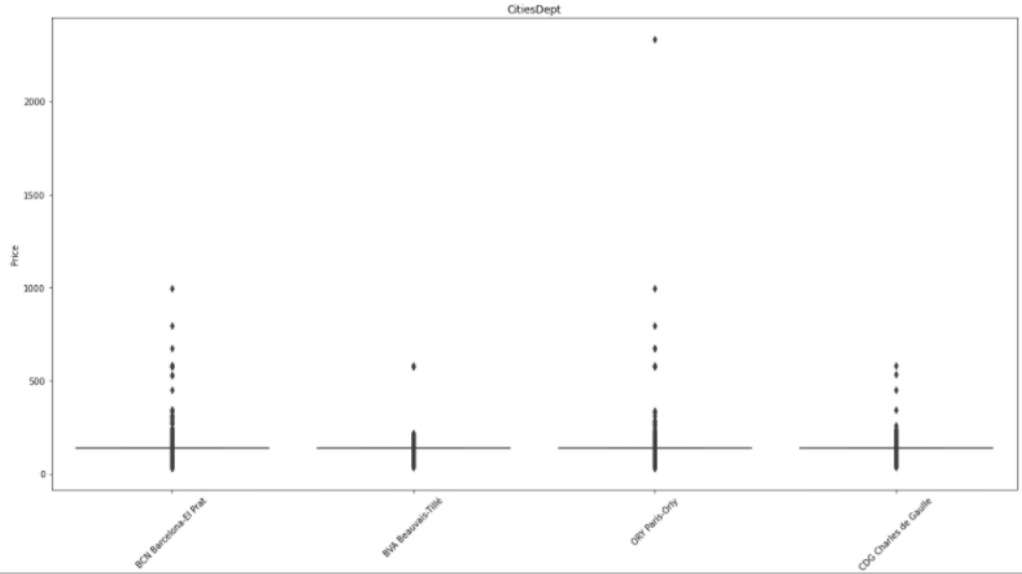


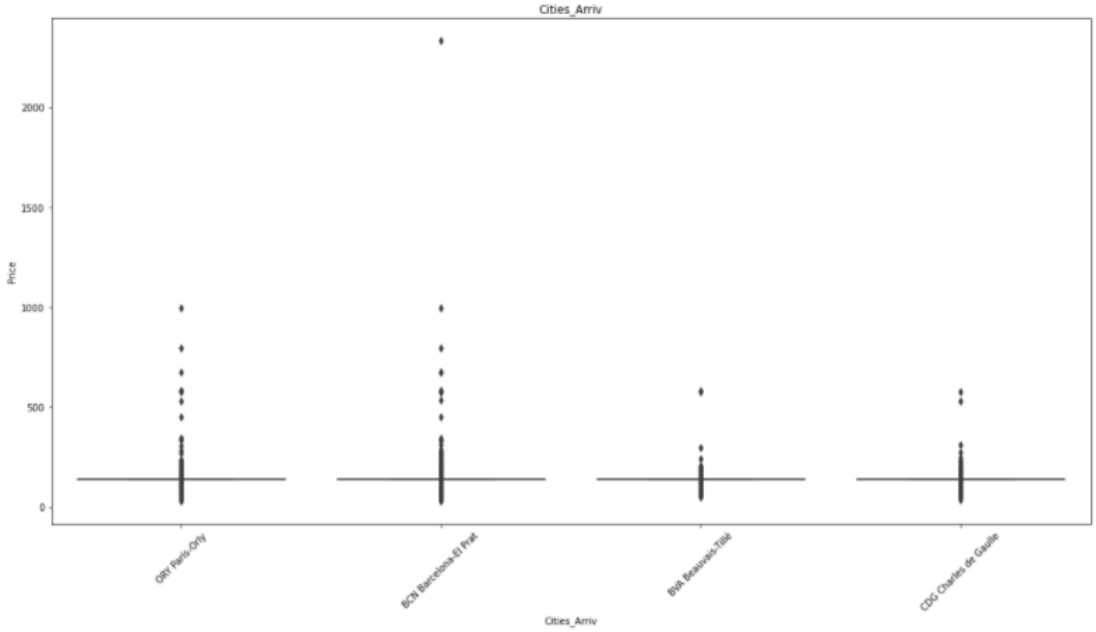
/srv/conda/envs/notebook/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only

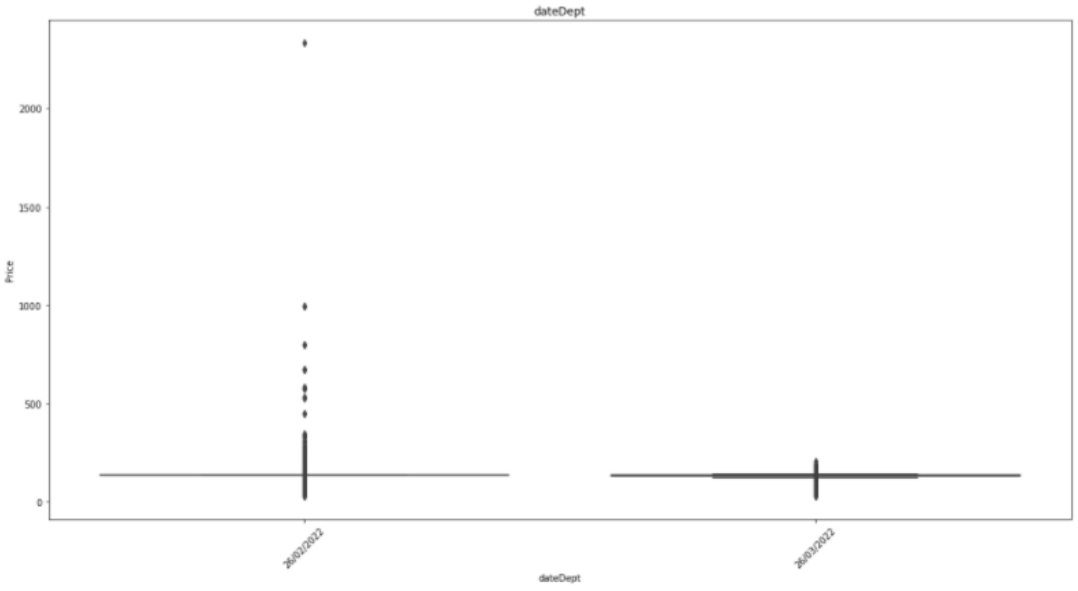


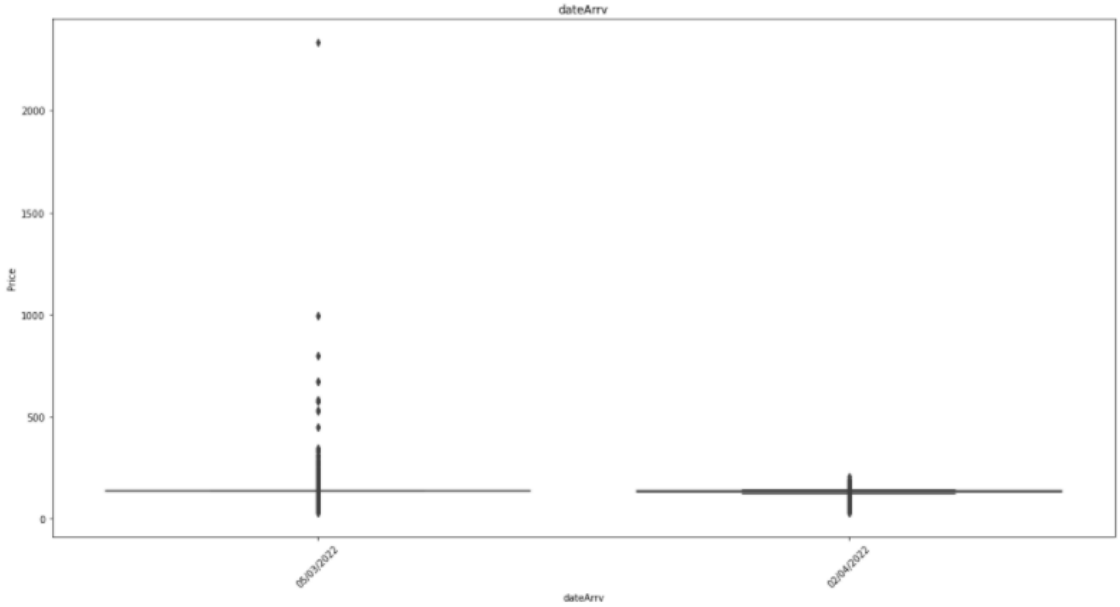

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only
```

FutureWarning

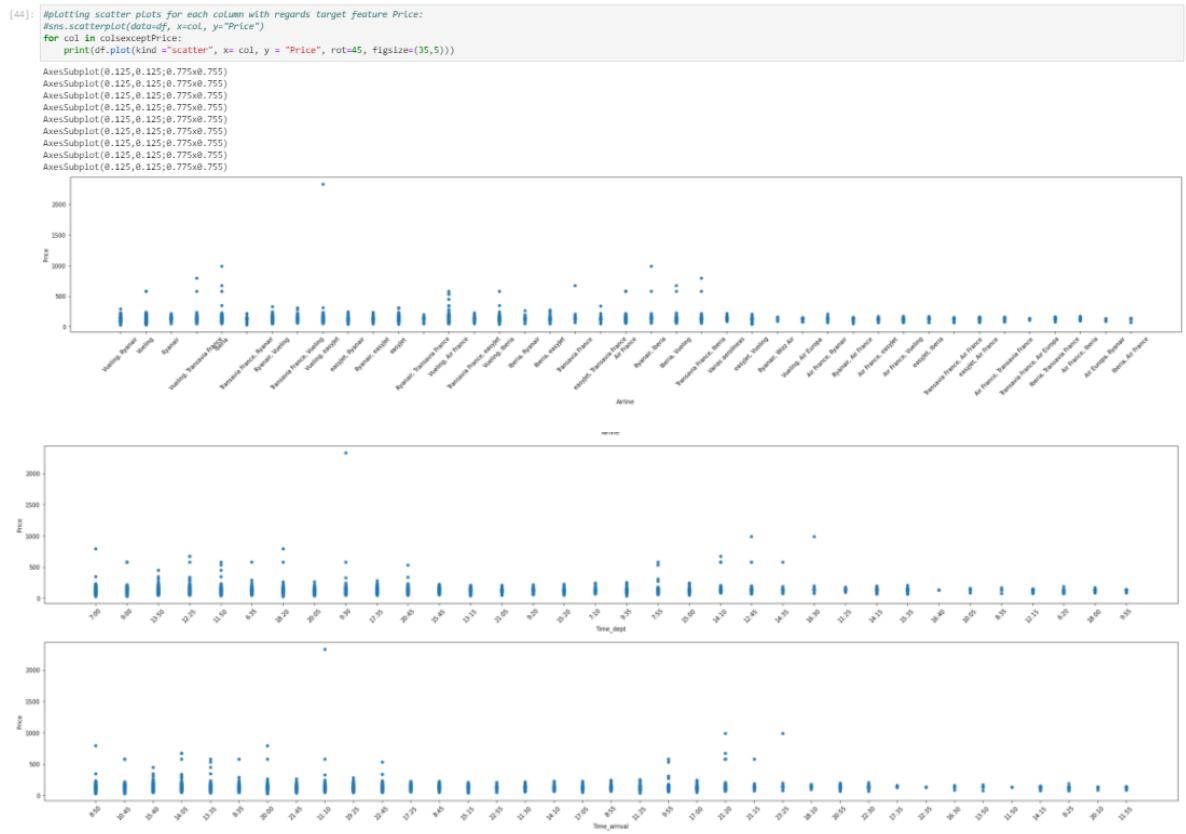


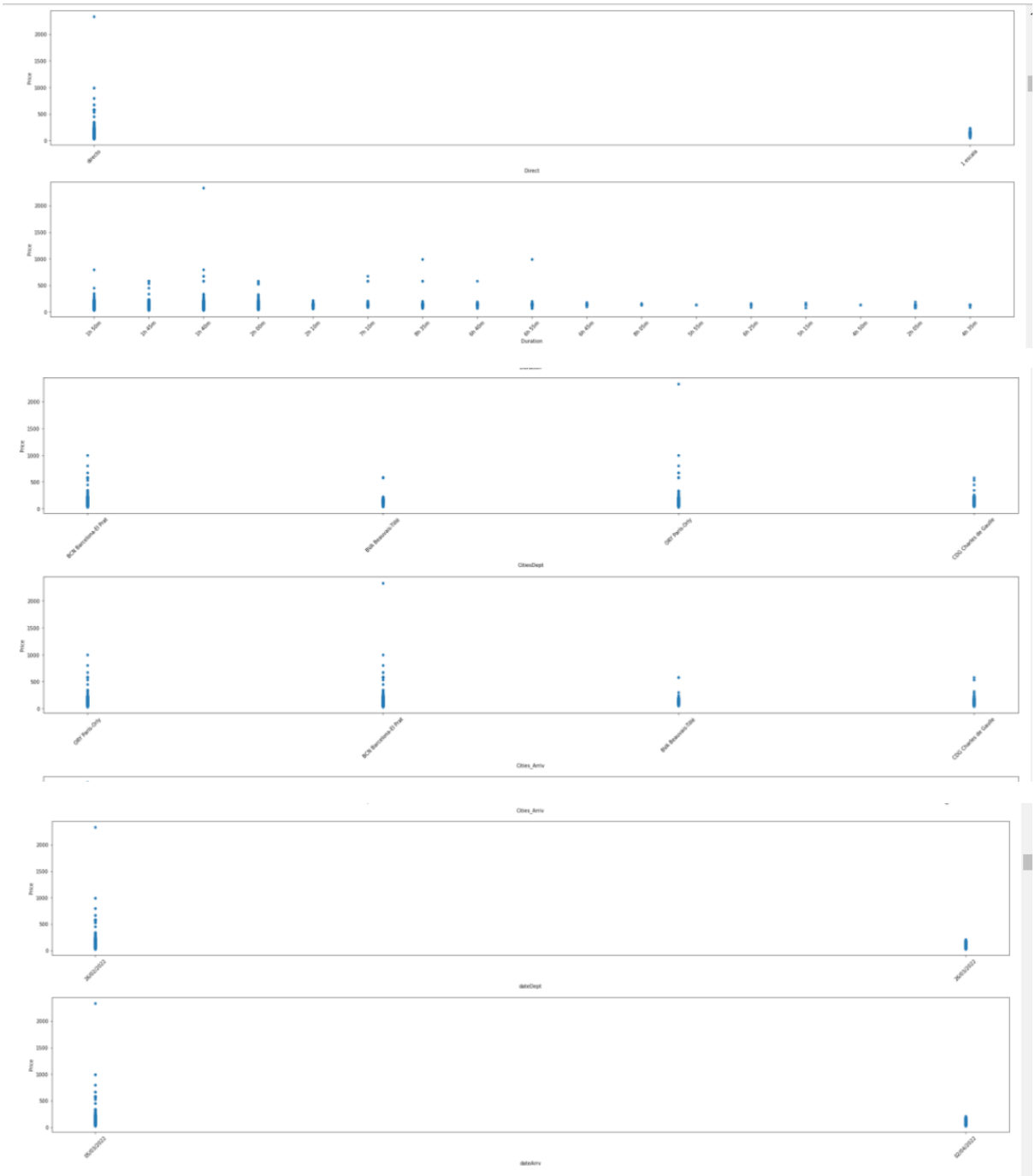

Cities_Arriv
Price
2000
1500
1000
500
0
ORY Paris_Orly
BCN Barcelona-El Prat
BVA Beauvais-Tillé
CDG Charles de Gaulle
Cities_Arriv


dateDept
Price
2000
1500
1000
500
0
26/02/2022
26/03/2022
dateDept

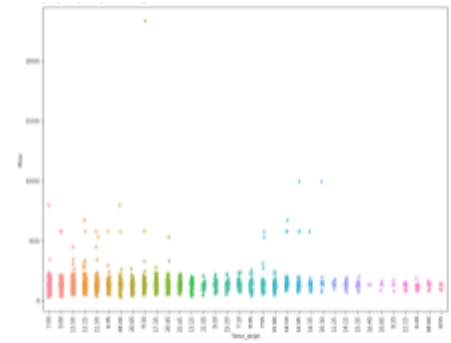


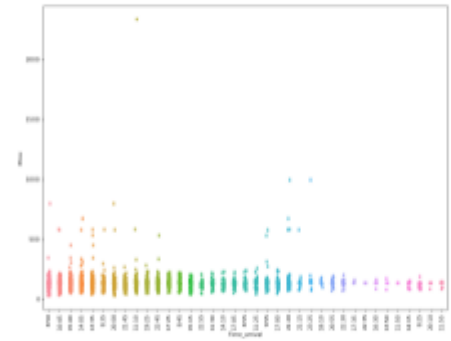
Here we can see the same conclusions as previously mentioned.

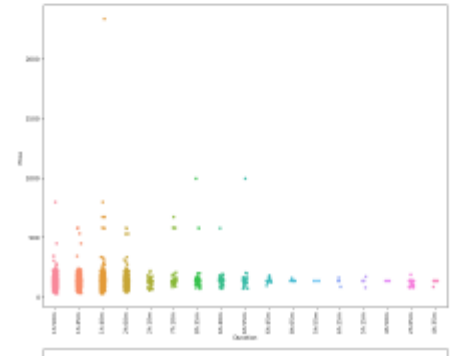


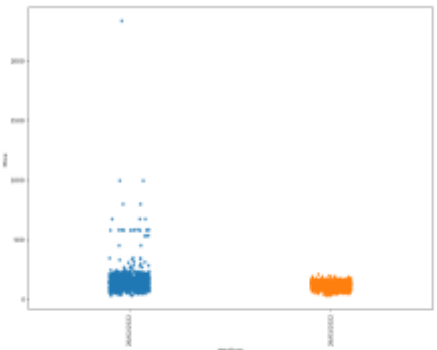


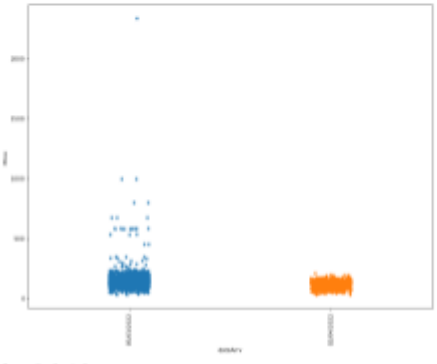
The same conclusion can be extrated from here, from the above scatter plots.











As we mentioned previously

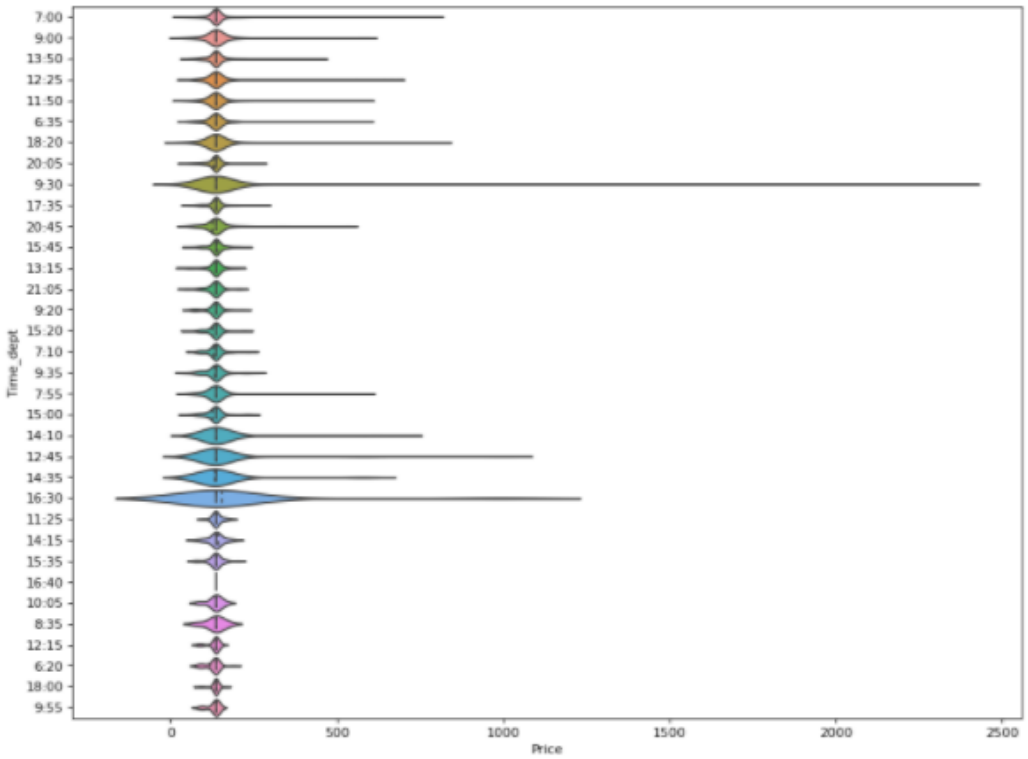
As we mentioned previously:

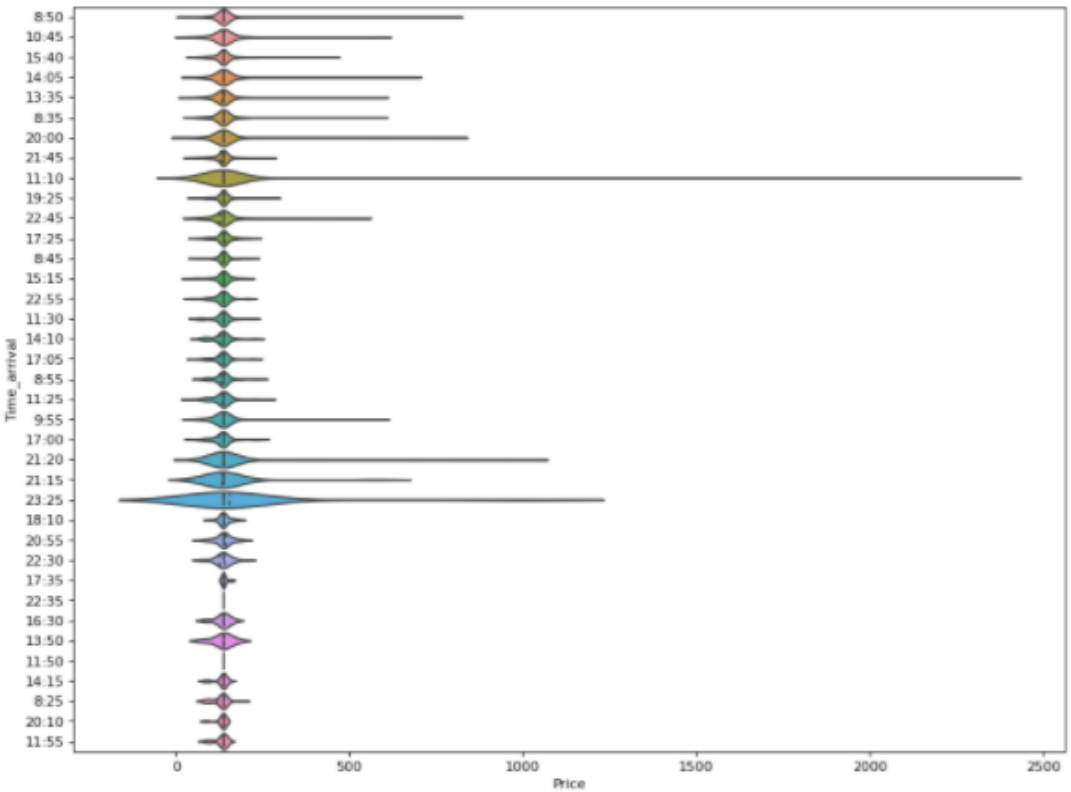
Regarding the departure time, we have few timings that may have higher prices than the mean and which are flights departing at 7am, 9am, 13:50pm, 12:25 pm, 11:50am, 18:20, 9:30am, 20:45, 7:55am, 14:10, 12:45pm, 14:35 and 16:30h.

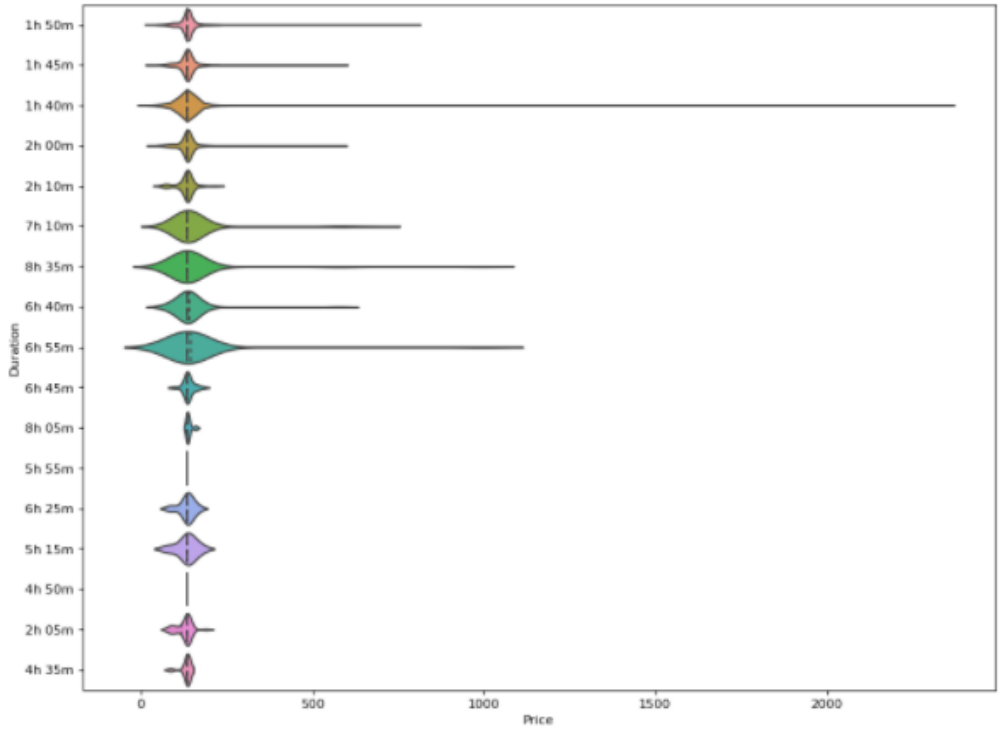
Similar situation happens with Arrival Time, we have Price around the mean and the median but we have also Arrival Time where we have a bit higher prices such as in timings like 8:50, 10:45 (one case), 15:40, 14:05, 13:35,8:35 (one case), 20h (2 cases), 21:45, 11:10, 22:45 (2 cases), 9:55, 21:20, 21:15 (one ase) and 23:25 (one case).

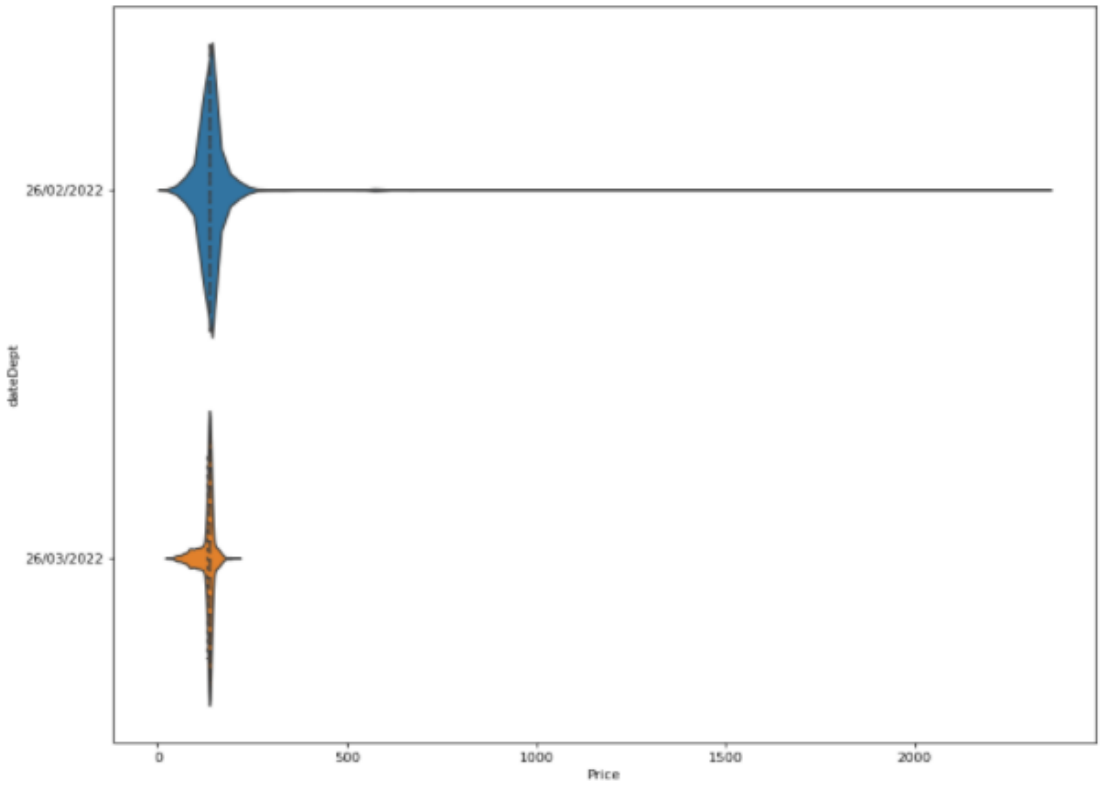
When comparing Duration, it can be studied and analyzed the same as tge departure and arrival time. Almost al the prices are between 0 and 500 euros except some cases where we have a bit higher prices such as duration of 1h 50min, 1h 45m, 1h 40m (we have 1 outlier here), 2h, 7h 10m,, 8h 35m, 6h 40m and 6h 55m.

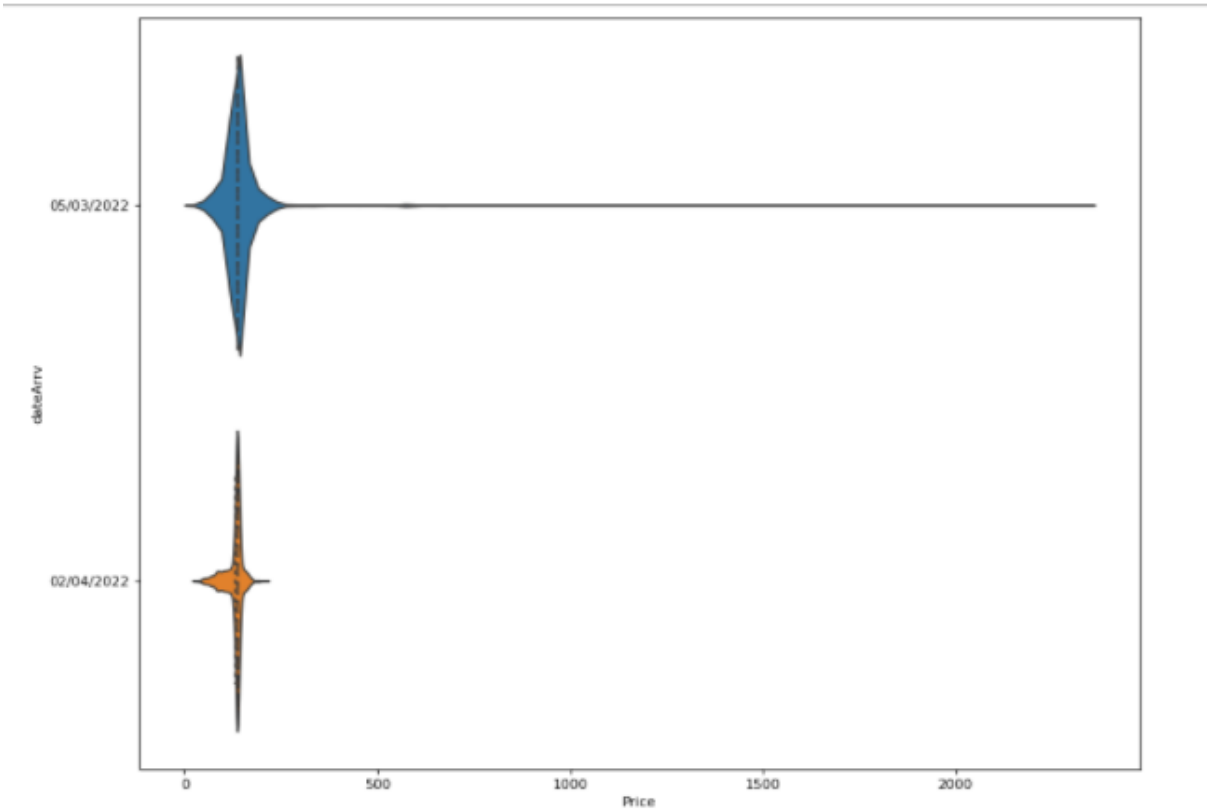
Regarding the dates, there is no big change in the Price when checking the flights for between dates 26 march and 2 April as it is just prior to Holy Wweek and the prices are down and close to the mean price while the prices between dates 26 february and 5 March tend to vary more than the other case.


Time_dept
7:00
9:00
13:50
12:25
11:50
6:35
18:20
20:05
9:30
17:35
20:45
15:45
13:15
21:05
9:20
15:20
7:10
9:35
7:55
15:00
14:10
12:45
14:35
16:30
11:25
14:15
15:35
16:40
10:05
8:35
12:15
6:20
18:00
9:55
0
500
1000
1500
2000
2500
Price


Time_arrival
8:50
10:45
15:40
14:05
13:35
8:35
20:00
21:45
11:10
19:25
22:45
17:25
8:45
15:15
22:55
11:30
14:10
17:05
8:55
11:25
9:55
17:00
21:20
21:15
23:25
18:10
20:55
22:30
17:35
22:35
16:30
13:50
11:50
14:15
8:25
20:10
11:55
0
500
1000
1500
2000
2500
Price

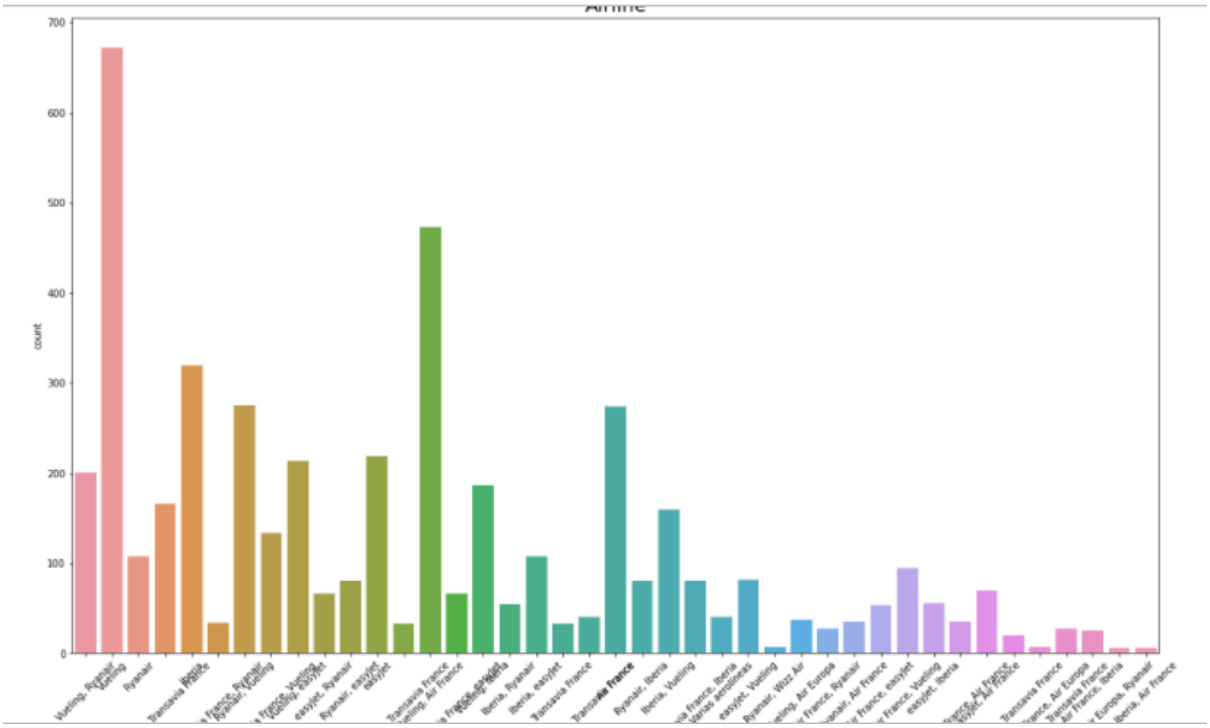

1h 50m
1h 45m
1h 40m
2h 00m
2h 10m
7h 10m
8h 35m
6h 40m
6h 55m
6h 45m
8h 05m
5h 55m
6h 25m
5h 15m
4h 50m
2h 05m
4h 35m
Duration
0
500
1000
1500
2000
Price

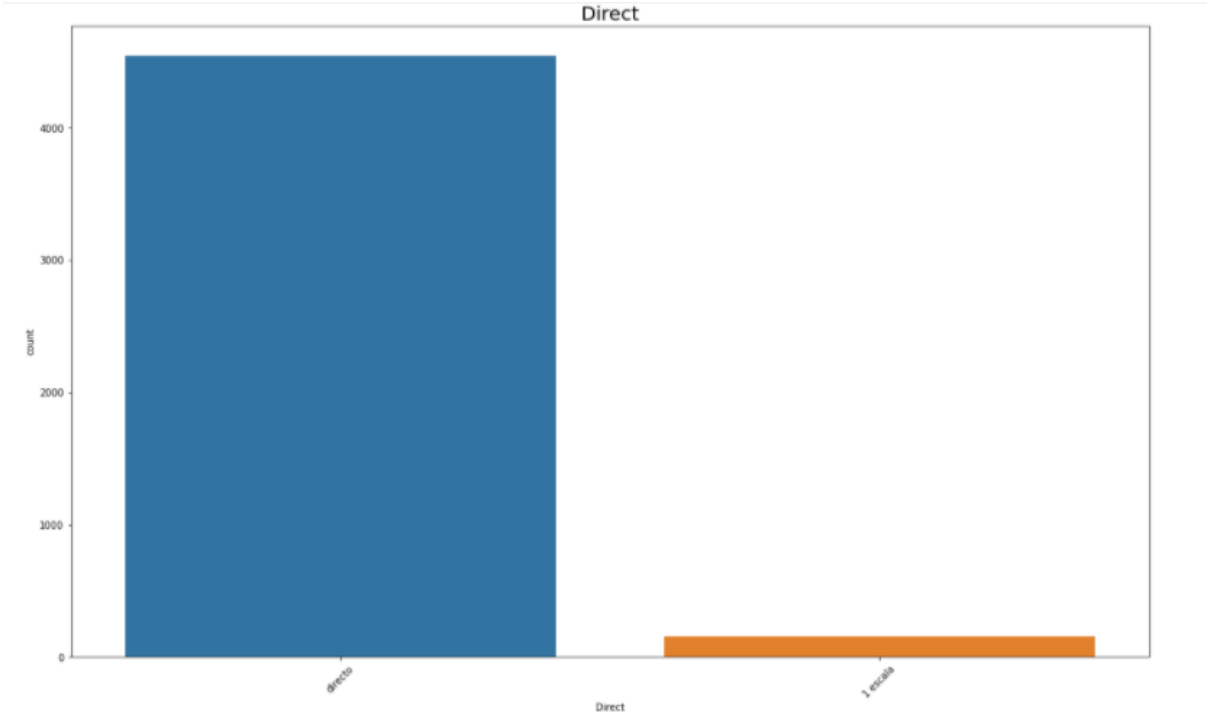

26/02/2022
26/03/2022
dateDept
0
500
1000
1500
2000
Price



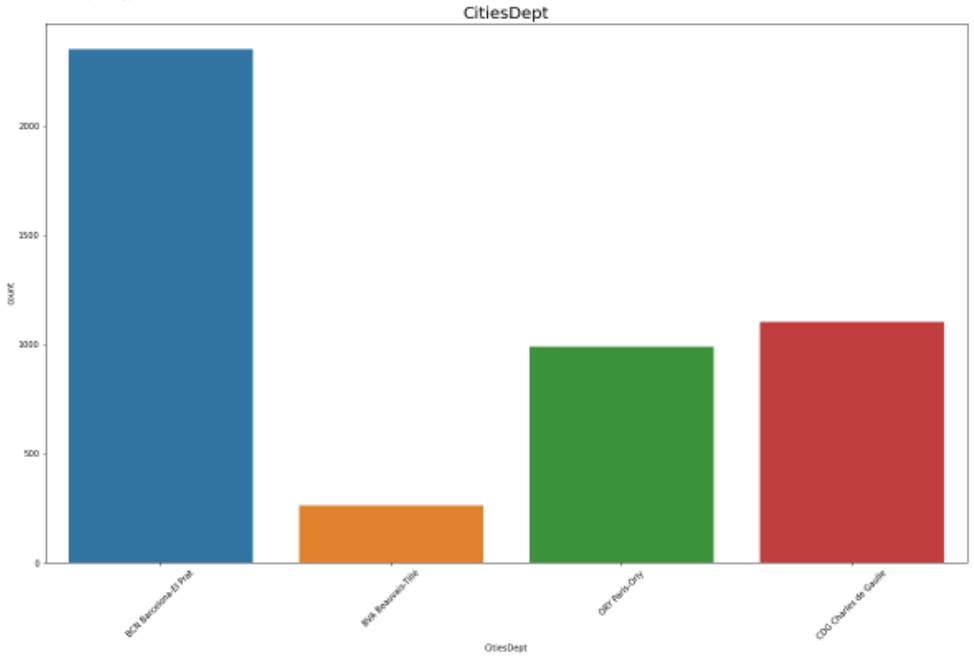
Here we can notice and conclude the same as previously commented.

As we said, the March Arrival date and departure date varies more on Price than the April dates prior to Holy week.

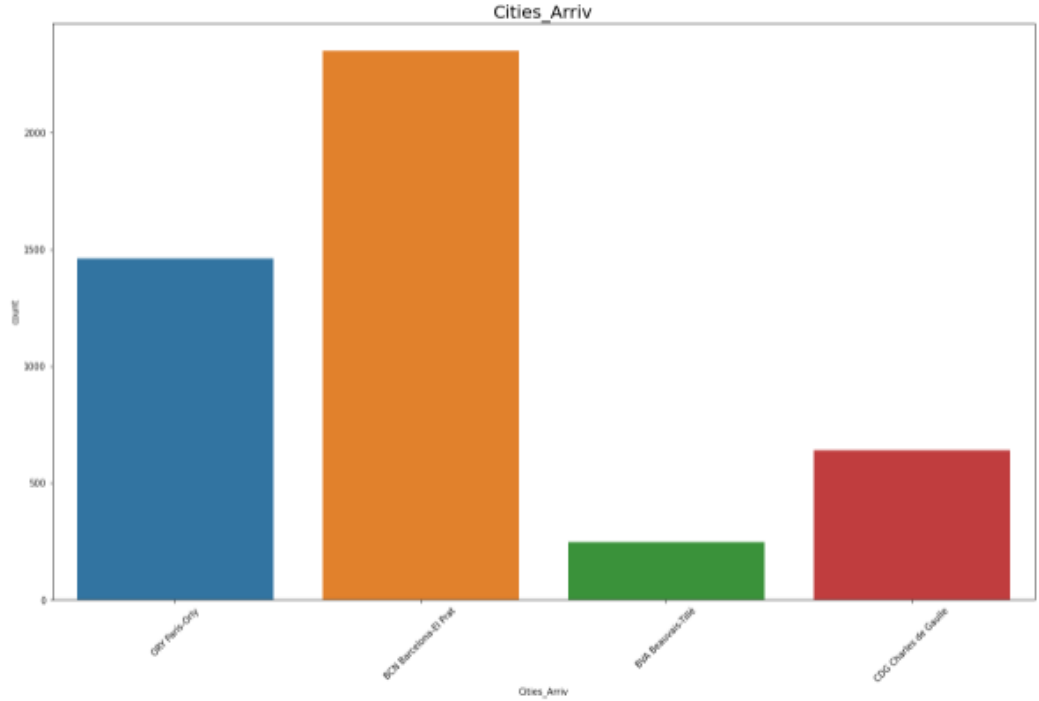

Airline
count
0
100
200
300
400
500
600
700


Direct
count
0
1000
2000
3000
4000
directo
1 escala
Direct

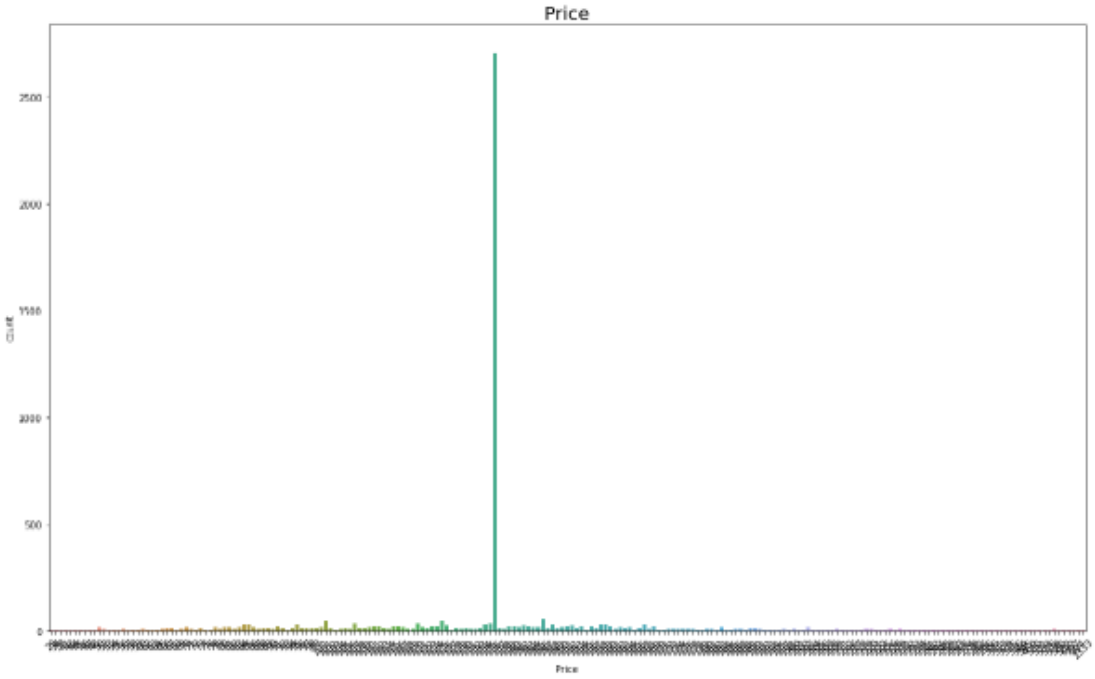
```
BCN Barcelona-El Prat    2351
CDG Charles de Gaulle    1101
ORY Paris-Orly            987
BVA Beauvais-Tillé        262
Name: CitiesDept, dtype: int64
```



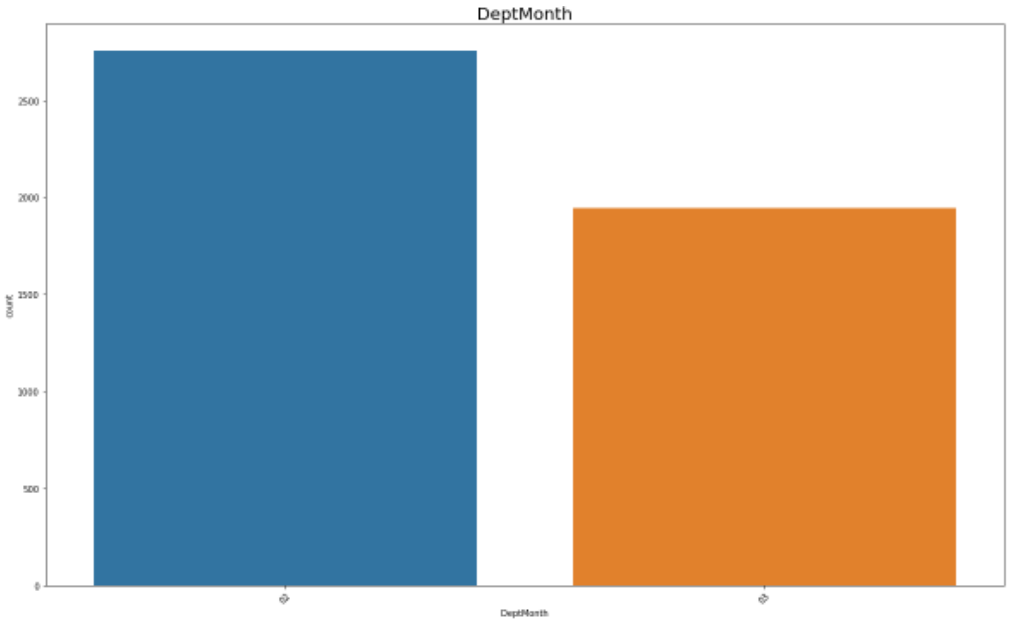

```
BCN Barcelona-El Prat    2350
ORY Paris-Orly           1462
CDG Charles de Gaulle     641
BVA Beauvais-Tillé        248
Name: Cities_Arriv, dtype: int64
```

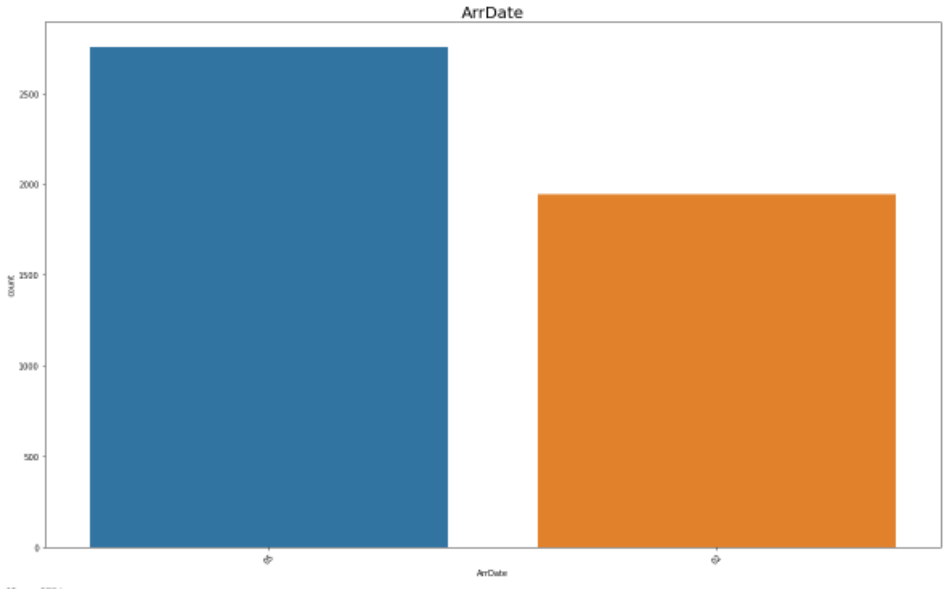


```
136    2784
146      56
101      45
125      45
107      34
       ...
274       1
278       1
282       1
286       1
296       1
Name: Price, Length: 215, dtype: int64
```

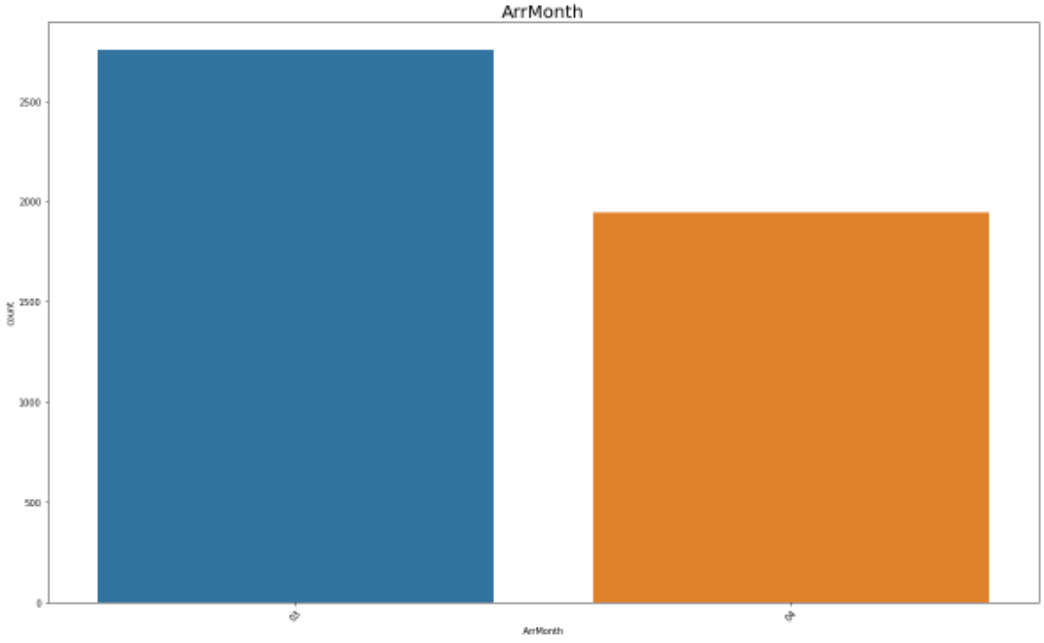


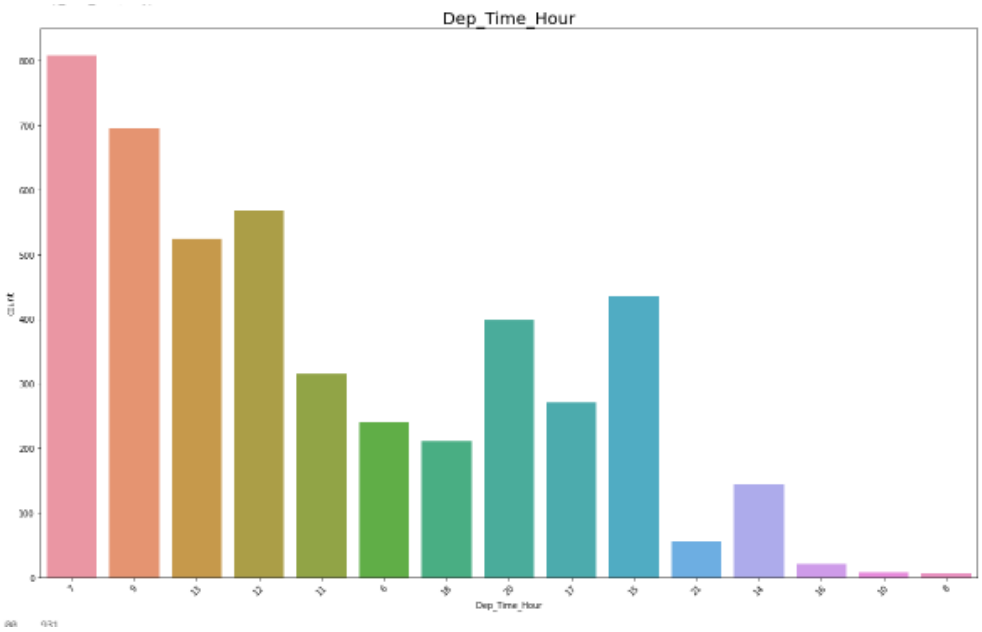

```
25    4701
```



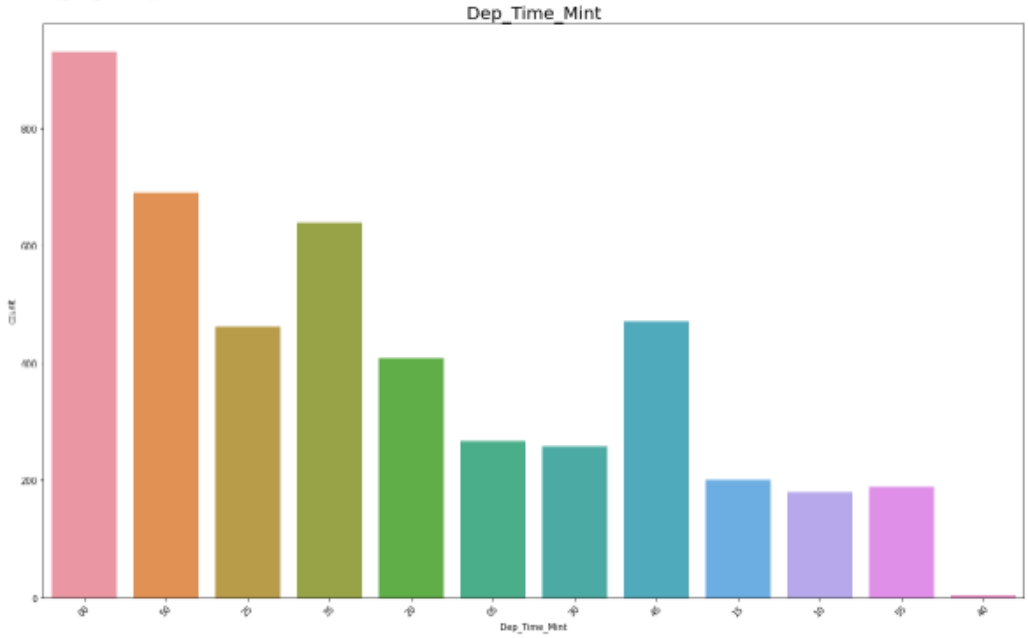



```
03    2754
04    1947
Name: ArrMonth, dtype: int64
```



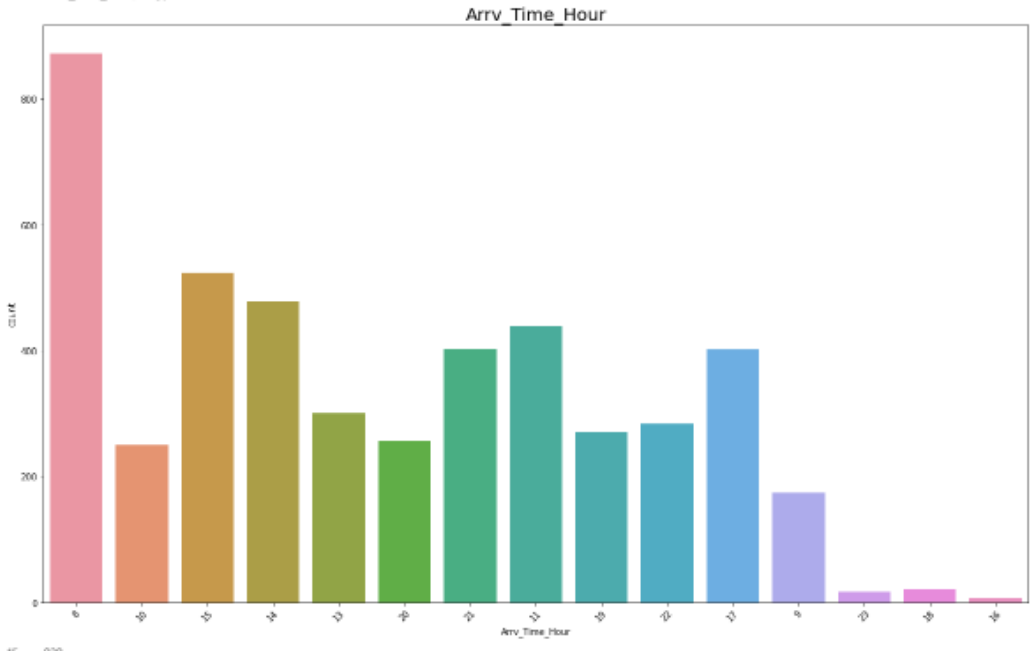



```
55    183
10    188
40      4
Name: Dep_Time_Mint, dtype: int64
```

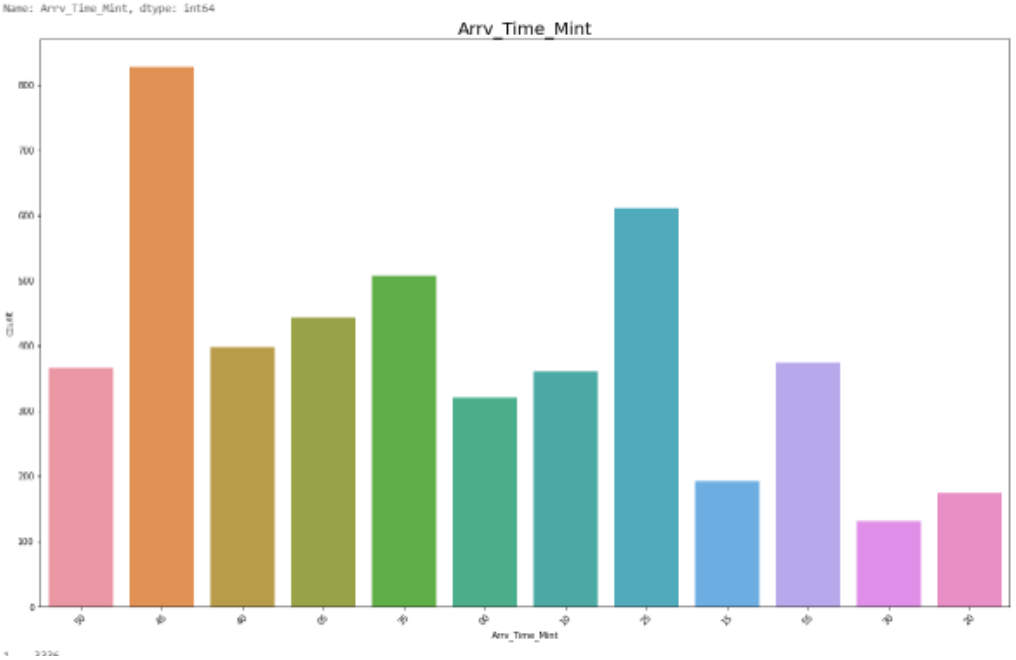



```
0     873
```

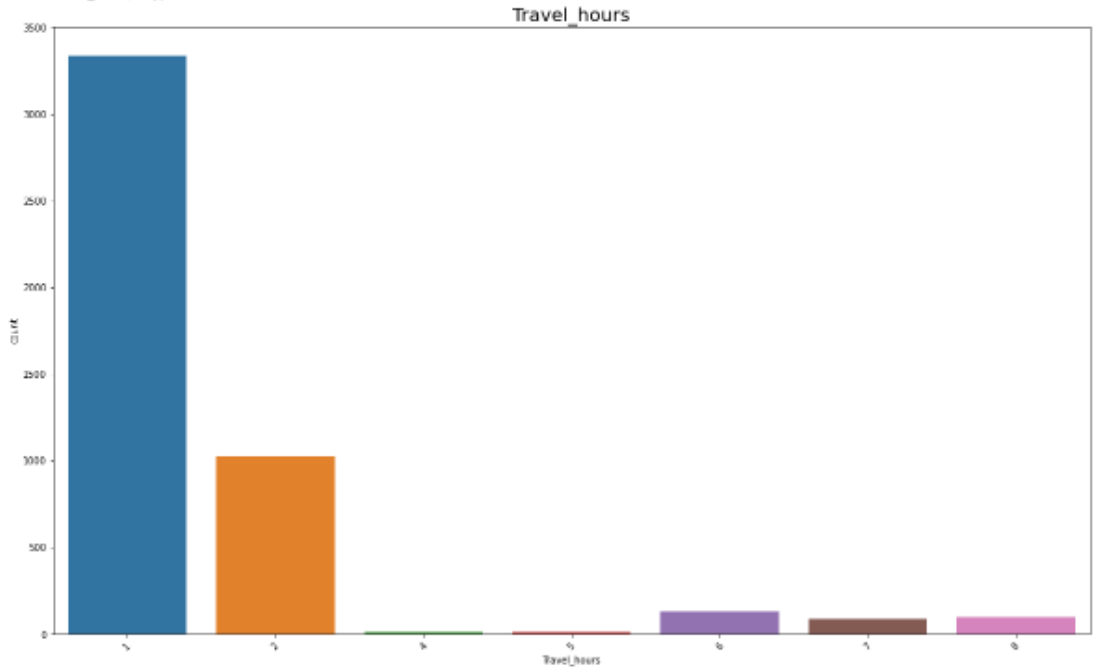
```
9     174
18     21
23     17
16      7
Name: Arrv_Time_Hour, dtype: int64
```



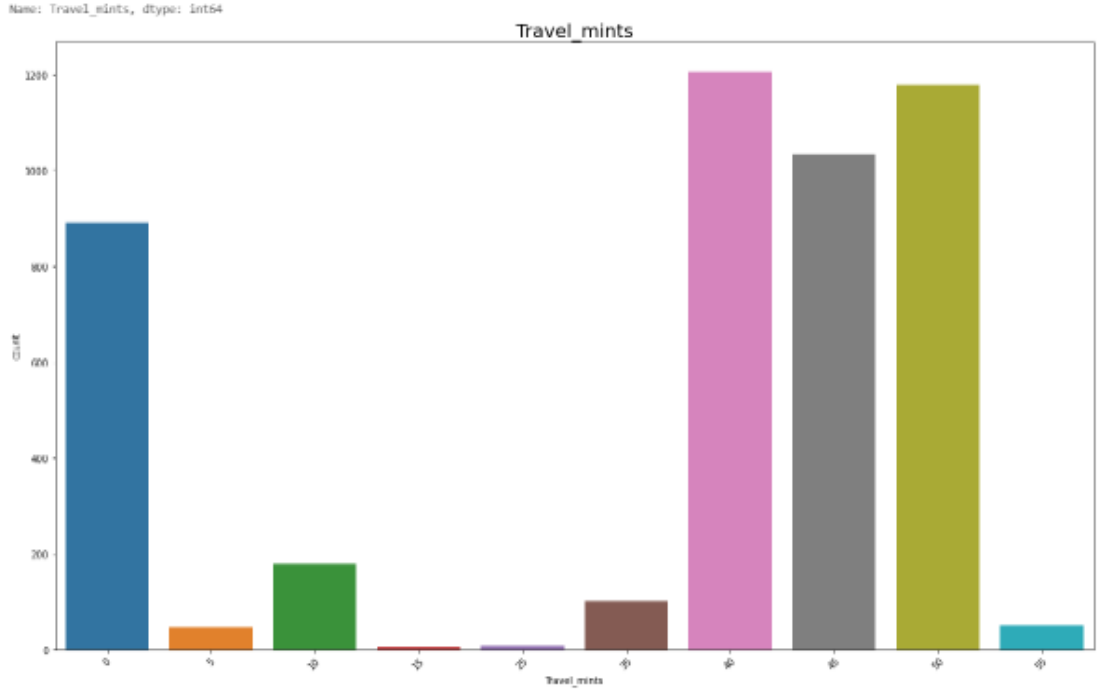

```
10    361
00    319
15    191
20    174
30    138
Name: Arrv_Time_Mint, dtype: int64
```



```
1    3336
2    1025
6     133
8      97
7      85
4      15
5      10
Name: Travel_hours, dtype: int64
```




```
50    1179
45    1033
0      892
10     179
35     101
55      58
5       47
25       7
15       6
Name: Travel_mints, dtype: int64
```



In the above barplots, we can see the following highlights:

- The airline with most flights offers is Vueling which top the list with 672 flights. Then, in second place, we have combo Vueling and Air France which offers 473 flights. Then, in third place, we have Iberia with 320 flights. And the flights combo Iberia and Air France or even Air Europa with combination Ryanair or even Transavia and Air Europa are the combos which have the lowest number of flights to offer.

- Regarding the direct flights, we have around 96,63% (4543) of the flights are direct flights and only 158 flights are fflights which have 1 stay in between flights.

- Barcelona airport is the airport where most airlines departs. And BVA Beauvais-Tille is the airport where less flights departs.

- Same happens with arrivals. Barcelona gets the most of the arrivals and BVA Beauvais-Tille gets the lowest numbers of arrivals.

- Regarding Price, we have range from 1 to 2704 euros.

- Departure day, we only have 1 day, which is 26th.

- Arrival day, we have 2 days, 5th and 2nd.

- Departure and Arrival month, we have 2 months, March and April.

- Departure and Arrival Year, only one, 2022.

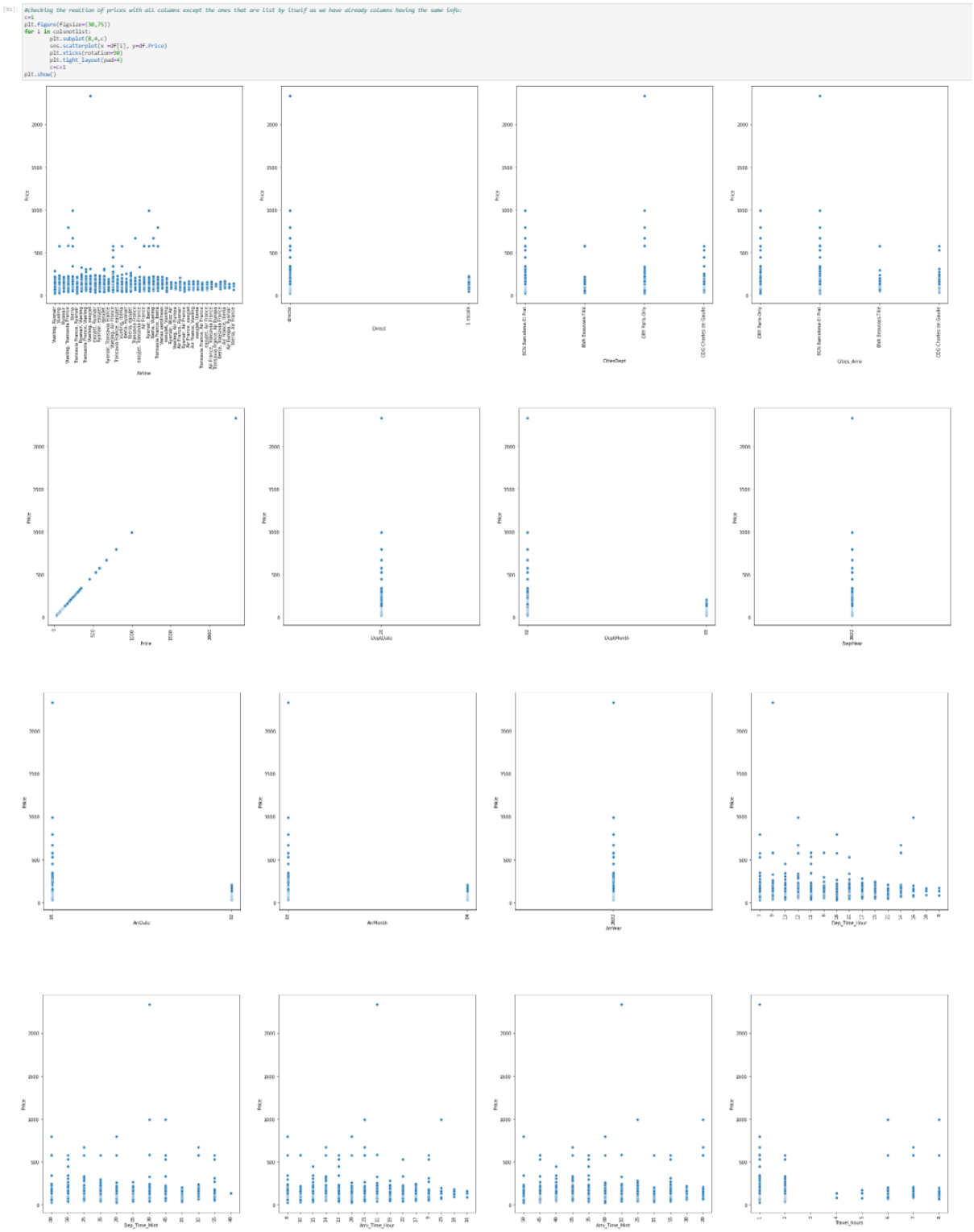
- regarding Departure Time Hours, we have range from 7am which has around 809 flights offer to 8h which has only 6 offers. We also have nights flights like 21h which has 56 flights.

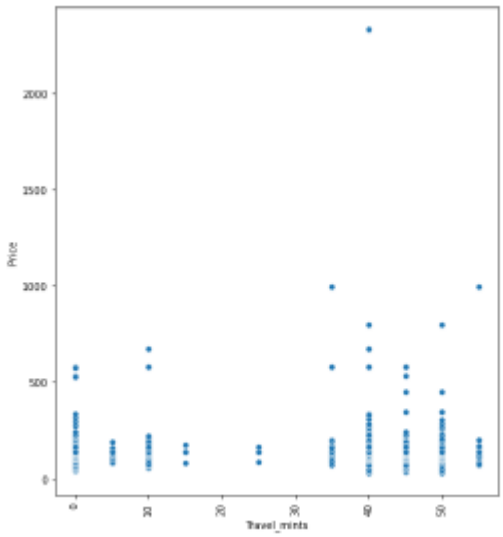
- regarding Departure Time Minutes, 00 minutes has the highest number of flights and 40minutes has the lowest number of flights.

- regarding Arrival Time Hours, we have range from 8am which has around 873 flights offer to 16h which has only 7 offers. We also have nights flights arrivals like 23h which has 17 flights.

- regarding Arrival Time Minutes, 45 minutes has the highest number of flights and 30 minutes has the lowest number of flights.

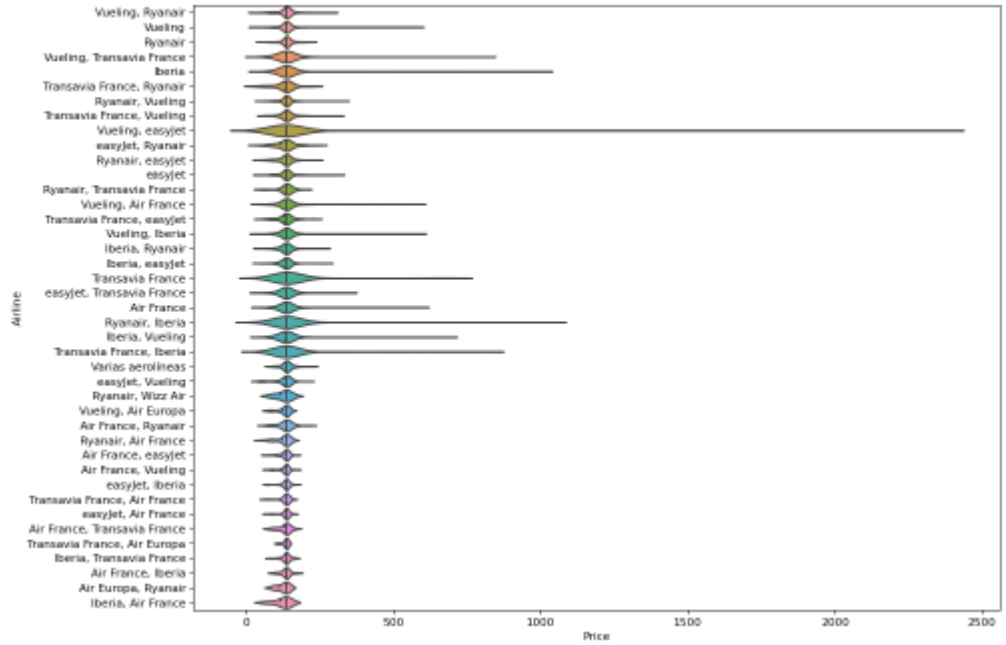
- mostly travel hours are between 1 to 2 hours and travel minutes are from 40 to 50 minutes.

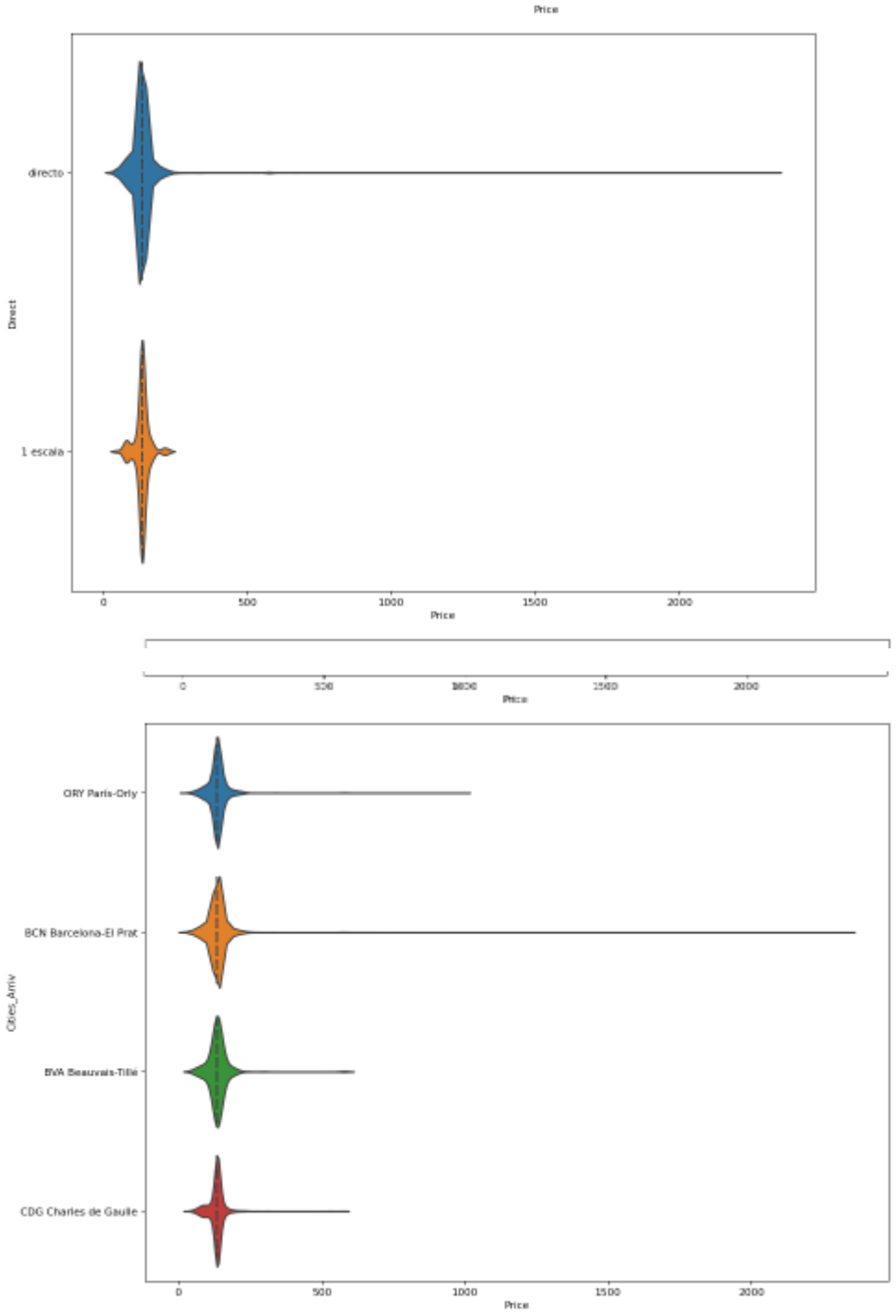


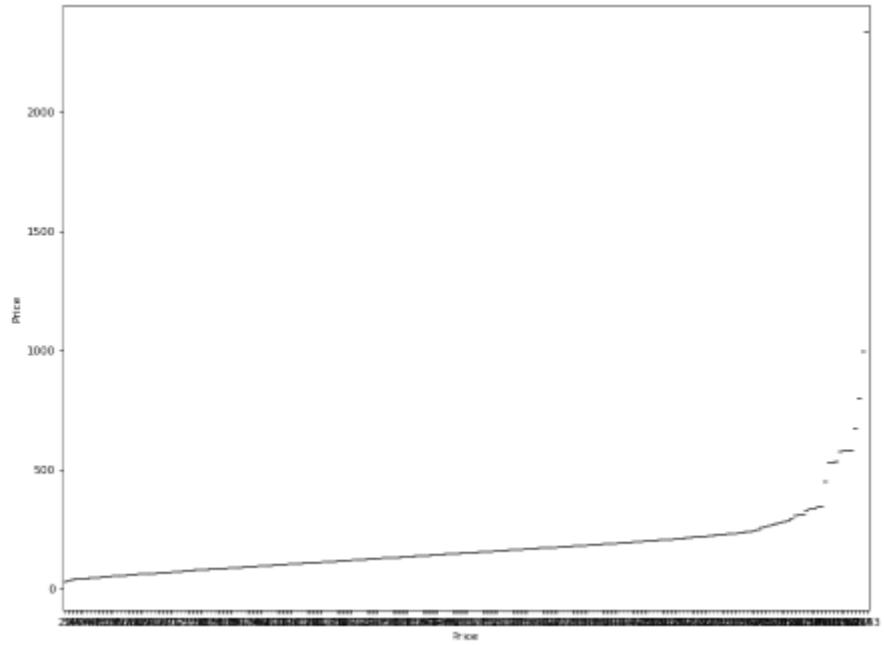


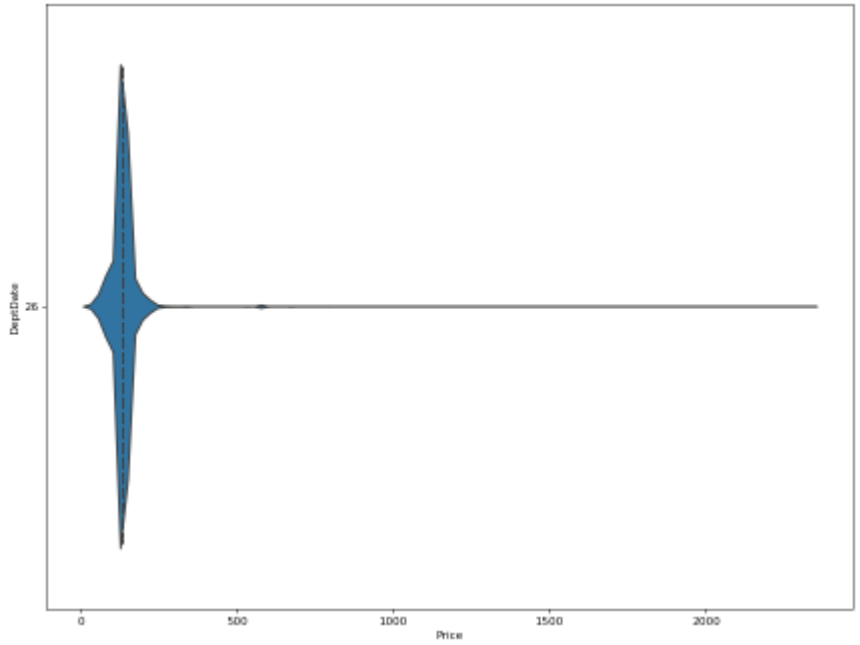
All these features work as categorical data as there is no linear relation between Price and other features.

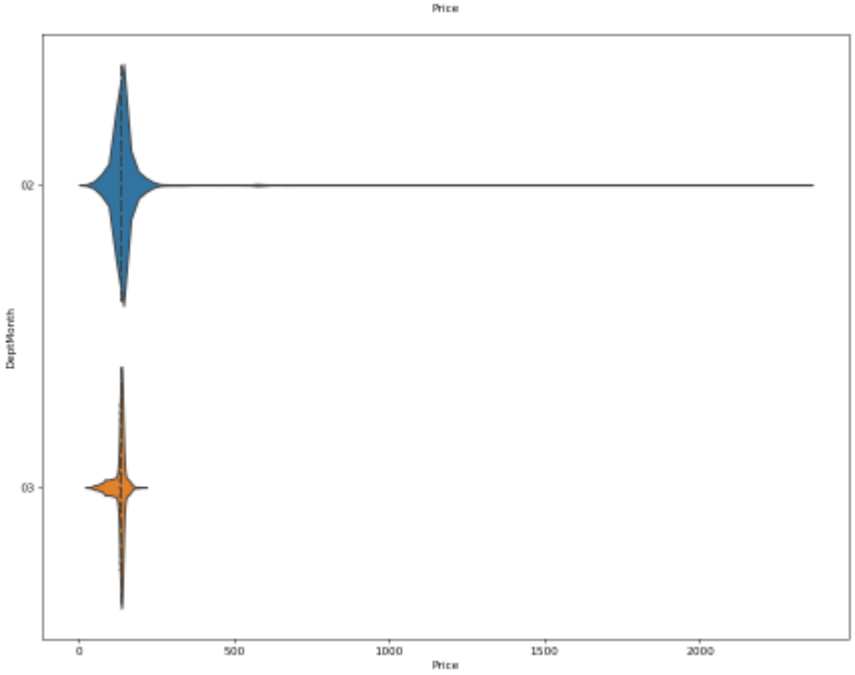
```
for col in colsnotlist:
    plt.figure(figsize=(13,10), dpi= 80)
    sns.violinplot(x='Price', y=col, data=df, scale='width', inner='quartile', rot=45)
```

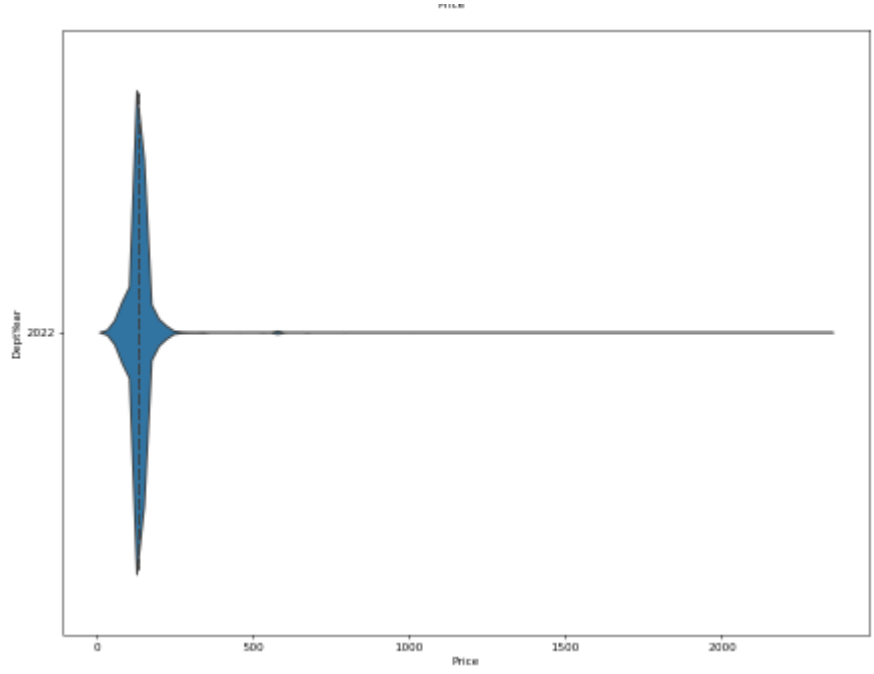


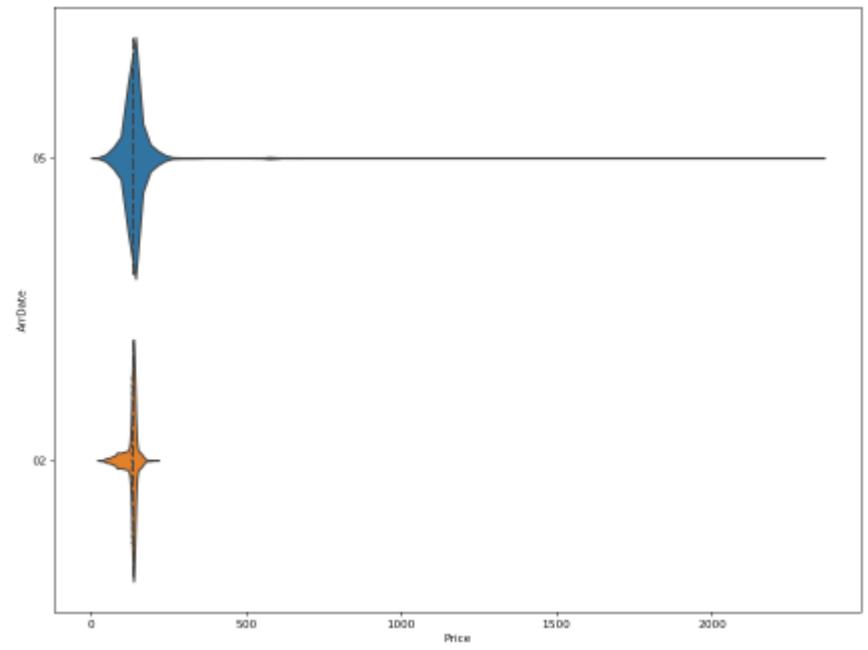

Price
directo
1 escala
Direct
0
500
1000
1500
2000
Price
ORY Paris-Orly
BCN Barcelona-El Prat
BVA Beauvais-Tille
CDG Charles de Gaulle
Cities_Arriv
0
500
1000
1500
2000
Price

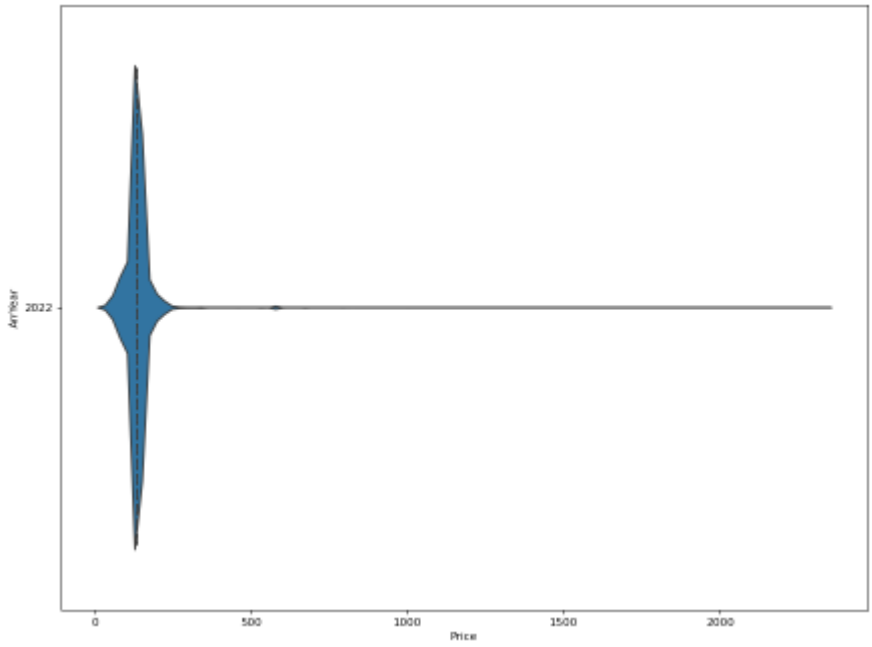

2000
1500
1000
500
0
Price
Price

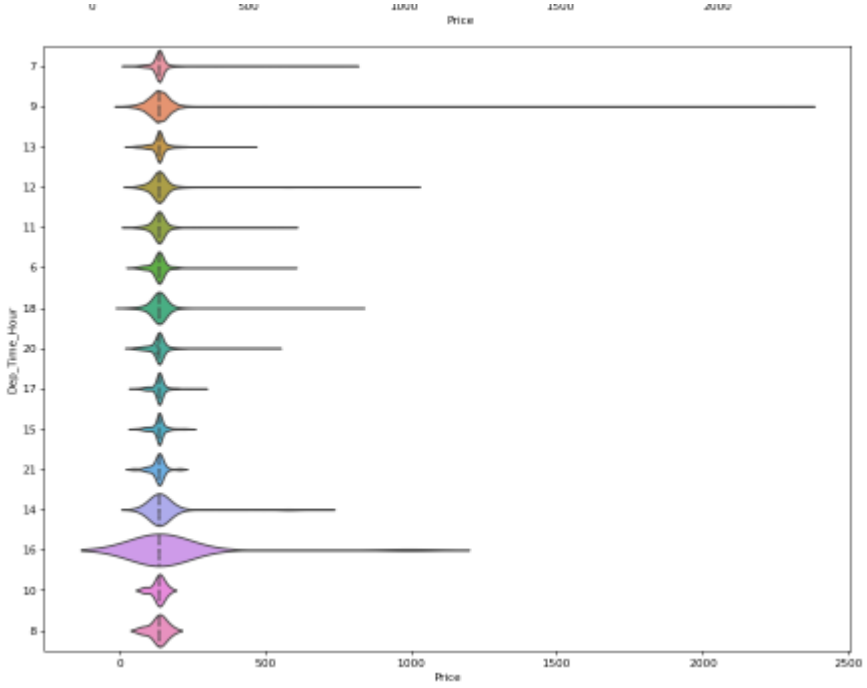

DeptDate
26
0
500
1000
1500
2000
Price

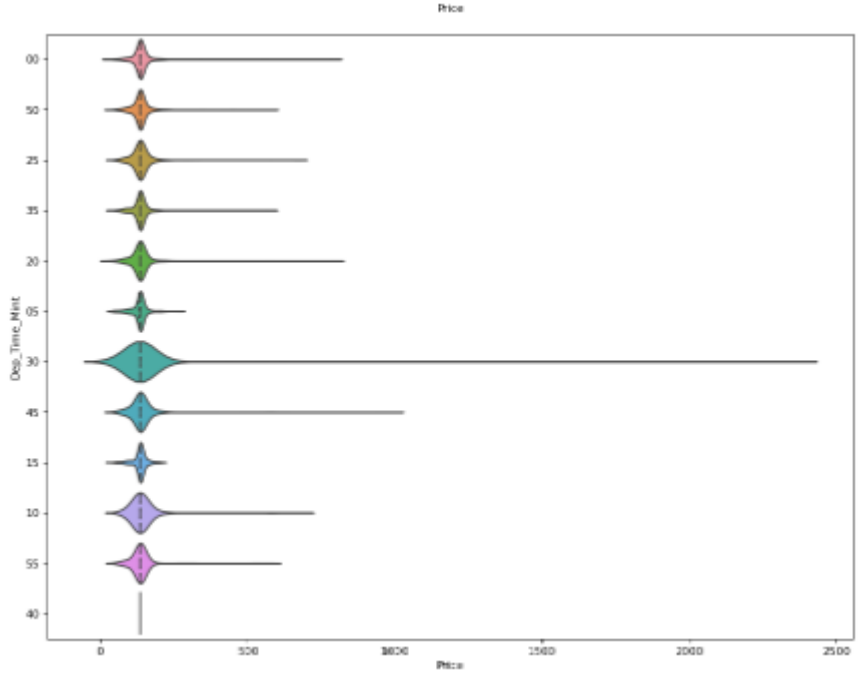

Price
02
03
DeptMonth
0
500
1000
1500
2000
Price

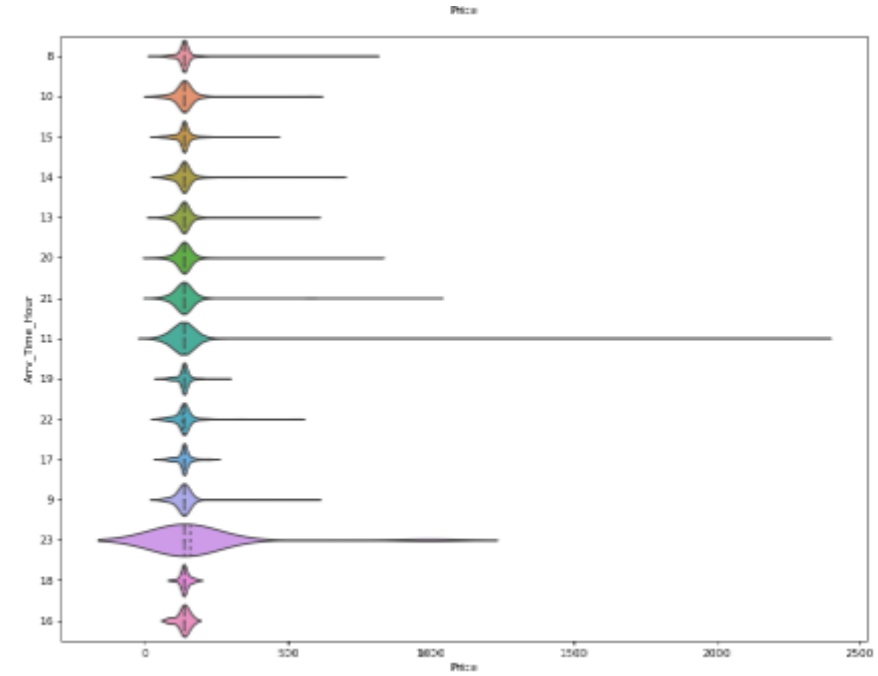

2022
DeptYear
0
500
1000
1500
2000
Price

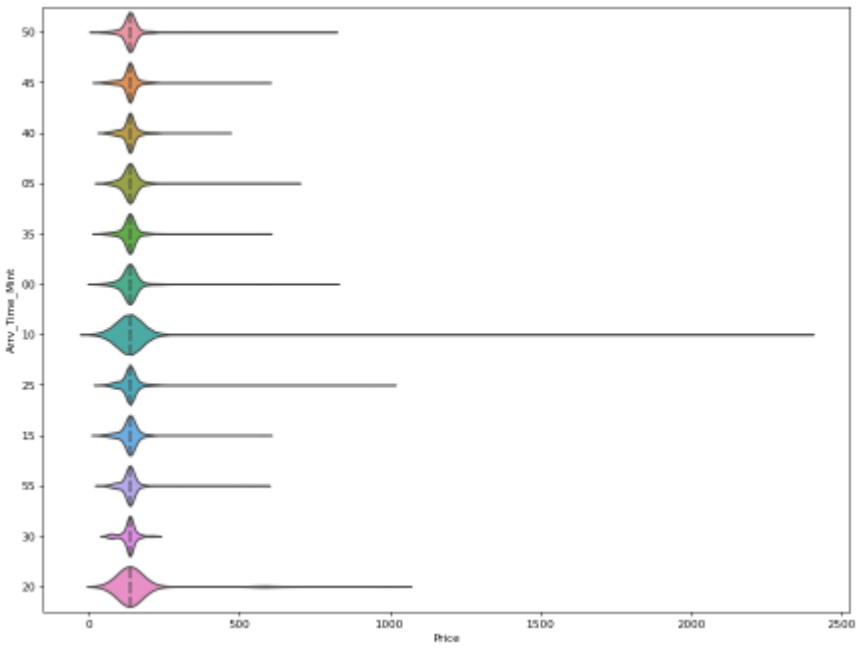

05
02
ArrDate
0
500
1000
1500
2000
Price

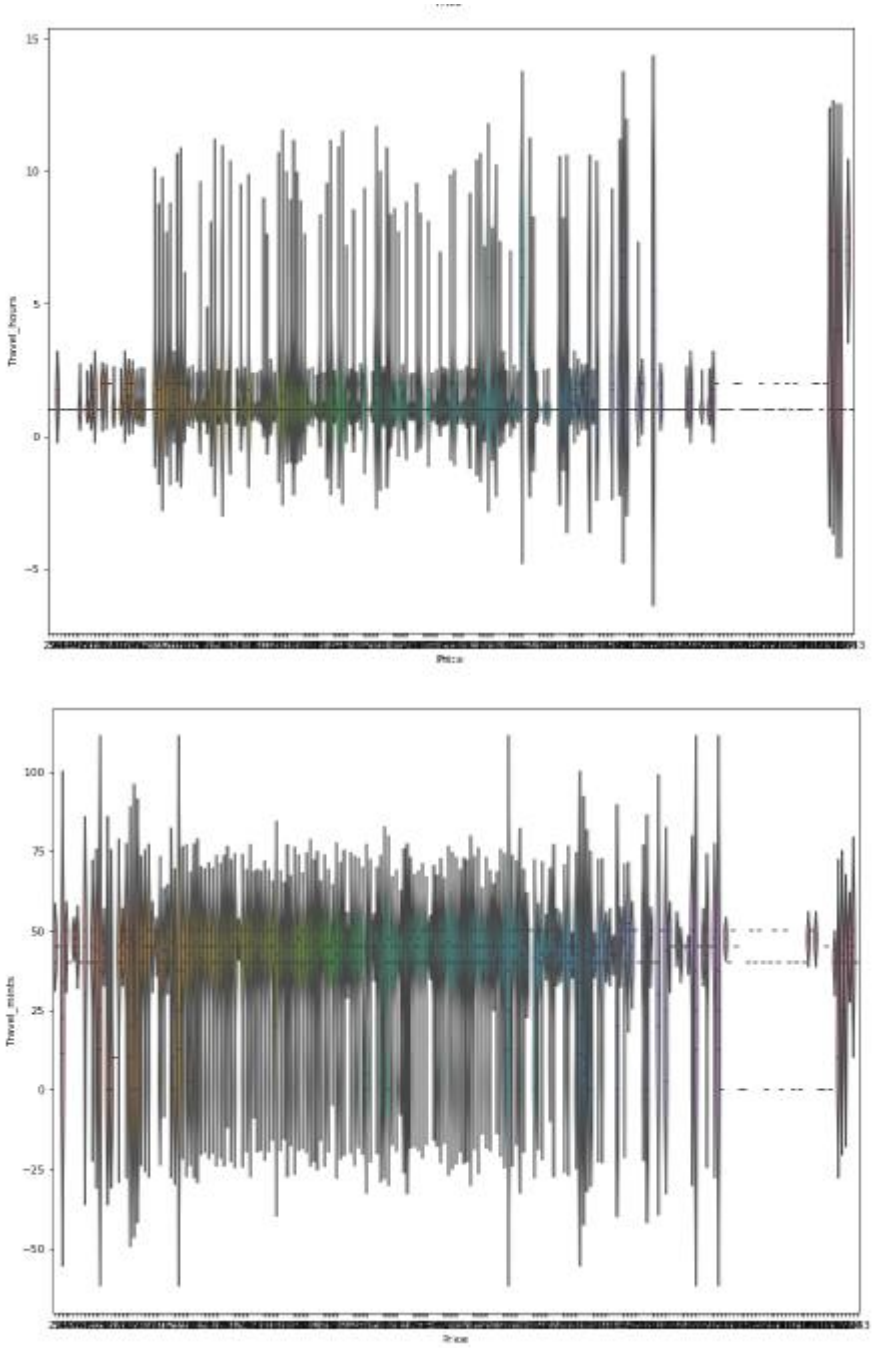

2022
ArrYear
0
500
1000
1500
2000
Price


Price
7
9
13
12
11
6
18
20
17
15
21
14
16
10
8
Dep_Time_Hour
0
500
1000
1500
2000
2500
Price


Price
00
50
25
35
20
05
30
45
15
10
55
40
Dep_Time_Mint
0
500
1000
1500
2000
2500
Price


Price
8
10
15
14
13
20
21
11
19
22
17
9
23
18
16
Arrv_Time_Hour
0
500
1000
1500
2000
2500
Price


50
45
40
05
35
00
10
25
15
55
30
20
Arrv_Time_Mint
0
500
1000
1500
2000
2500
Price



From the previous violinplots, we can highlight the following findings:

- Regarding the airlines, we have average flights prices between 0 and 250 euros.

- Even if we check the remaining features, we can conclude the same, the prices remain between 0 and 250 euros.

- If we check direct flights, we have more cases than the flights with one Stop/Stay.

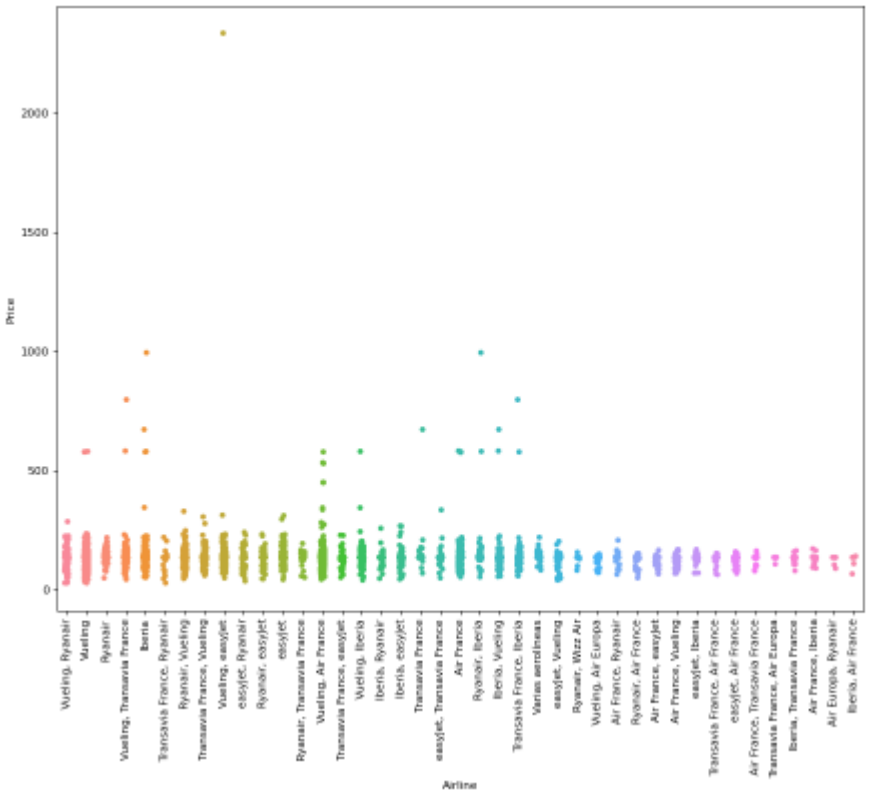
- Regarding the cities of departure and arrival in relation with price variation, we do not see big change in price variation over the cities of departure and arrival. We have similar violins,almost same, over the different options of cities of departure and arrival.

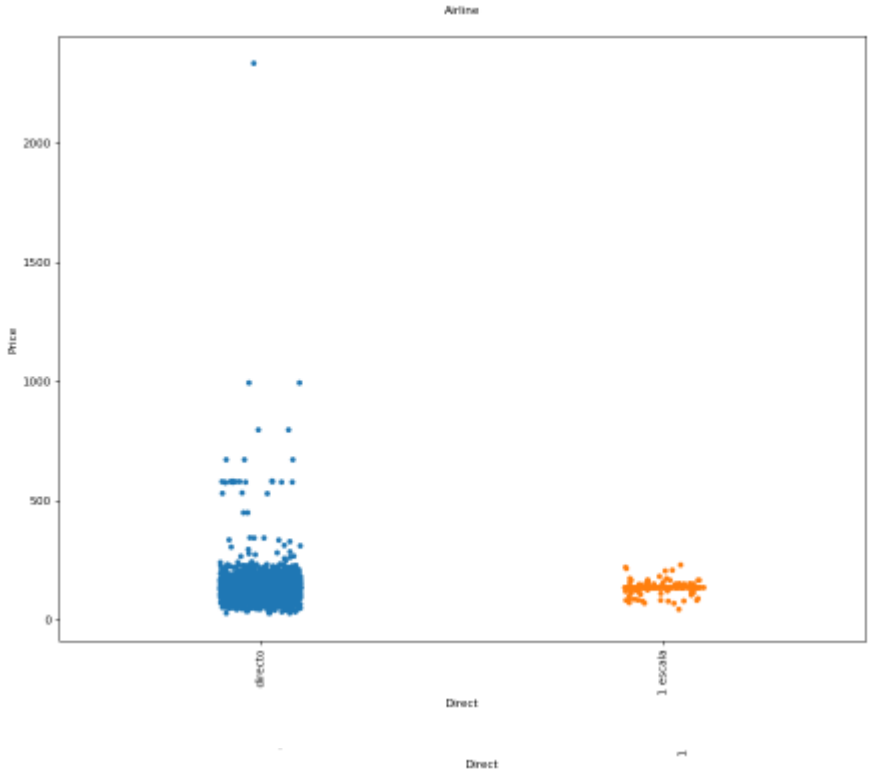
- Dept date day is the same for all cases 26th (Feb and March). And same happens with Year!

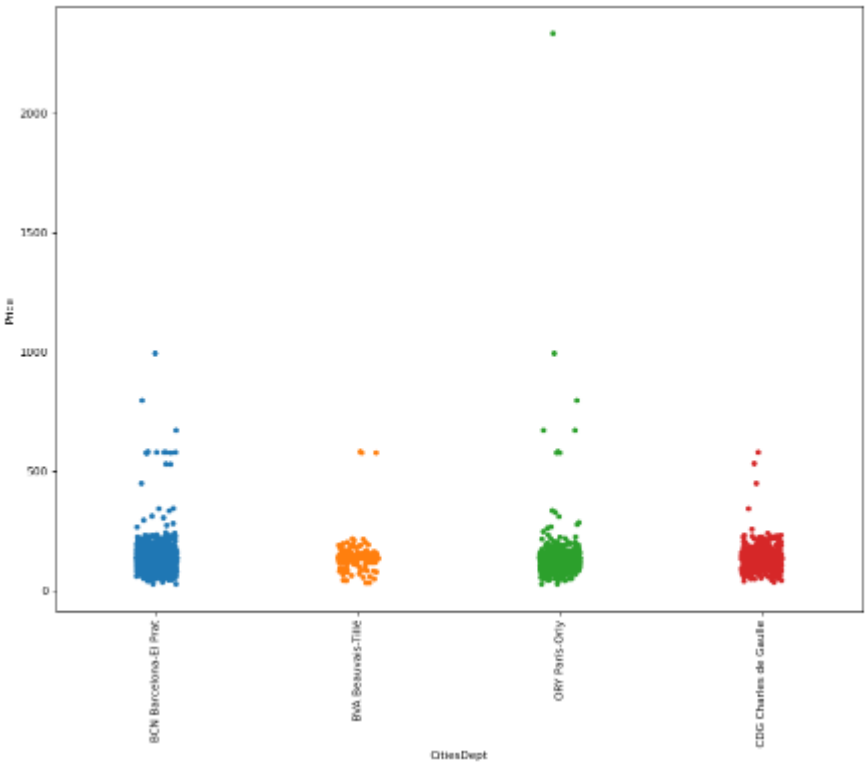
- Depart and Arival month are March and April. More flights in February than March as the flights that are leaving soon have more probability to be offered more than flights are away far.

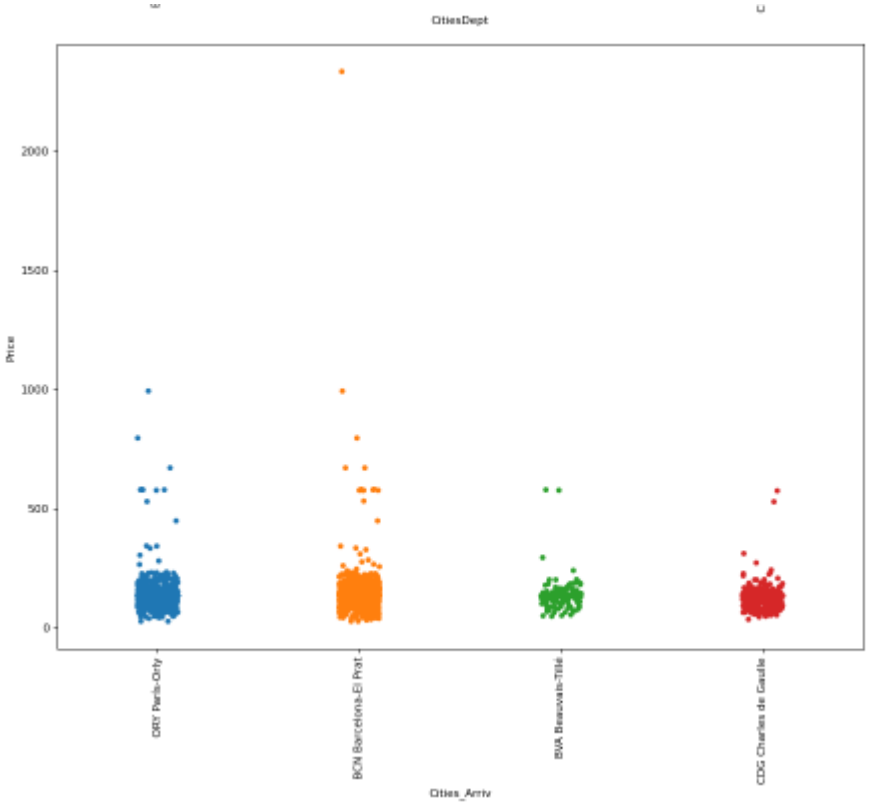
- Okay, if we check Depature Time Hour, we can say 16h has more variation than other in the Price standard deviation. Also 9am and 14h has some kind of variation on the Price deviation.

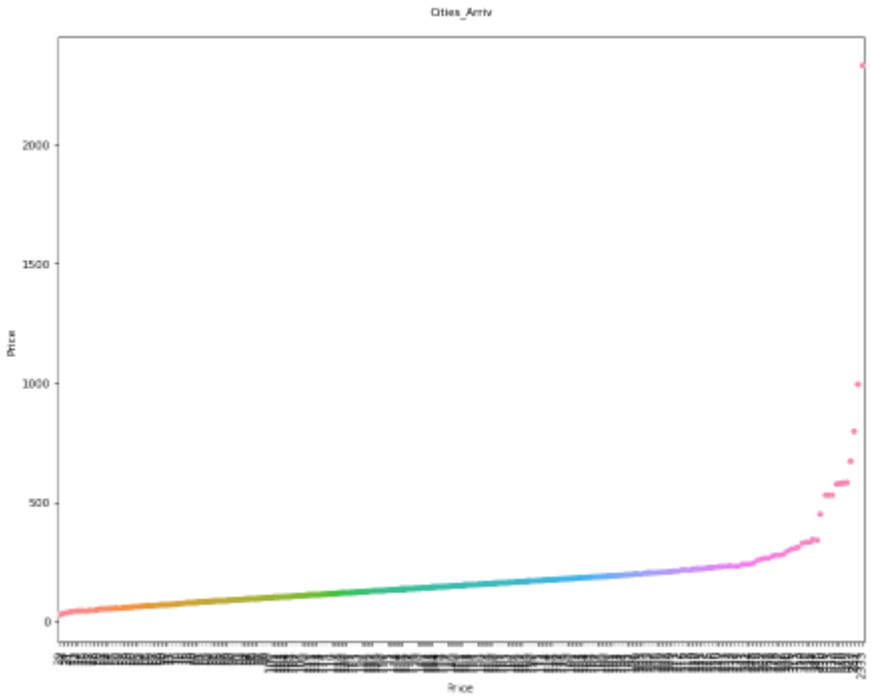
- Regarding Arrival Time in Hours, we see we have huge deviation in Price when arrival time hour is 23h.

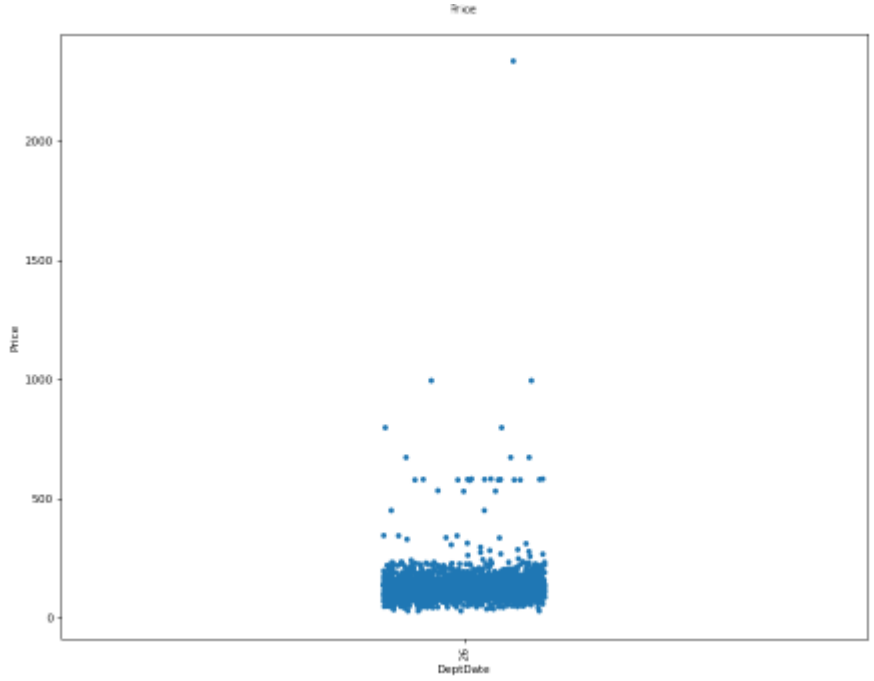


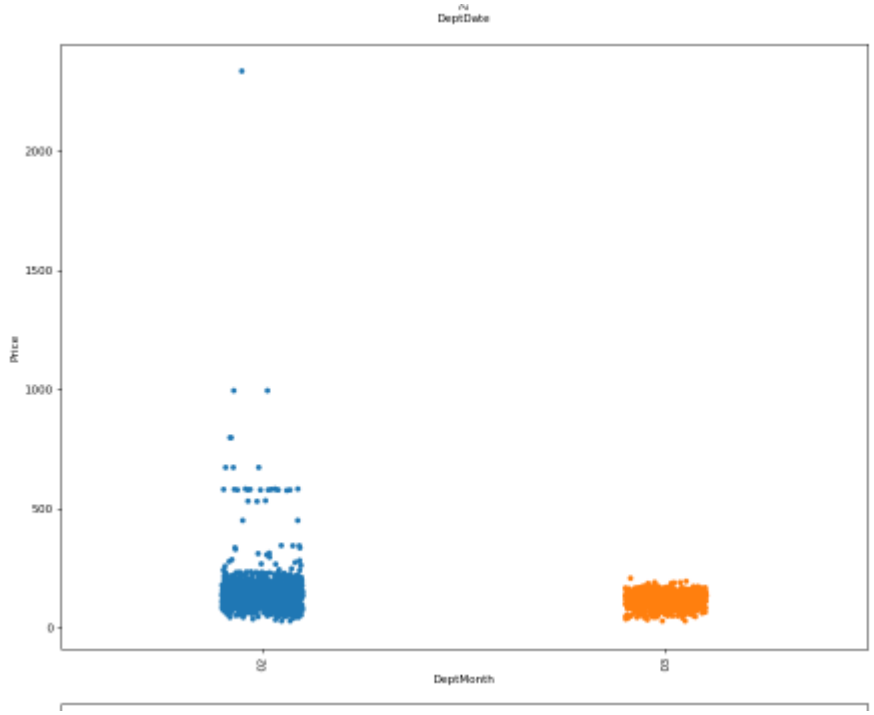

Airline
Price
2000
1500
1000
500
0
directo
1 escala
Direct

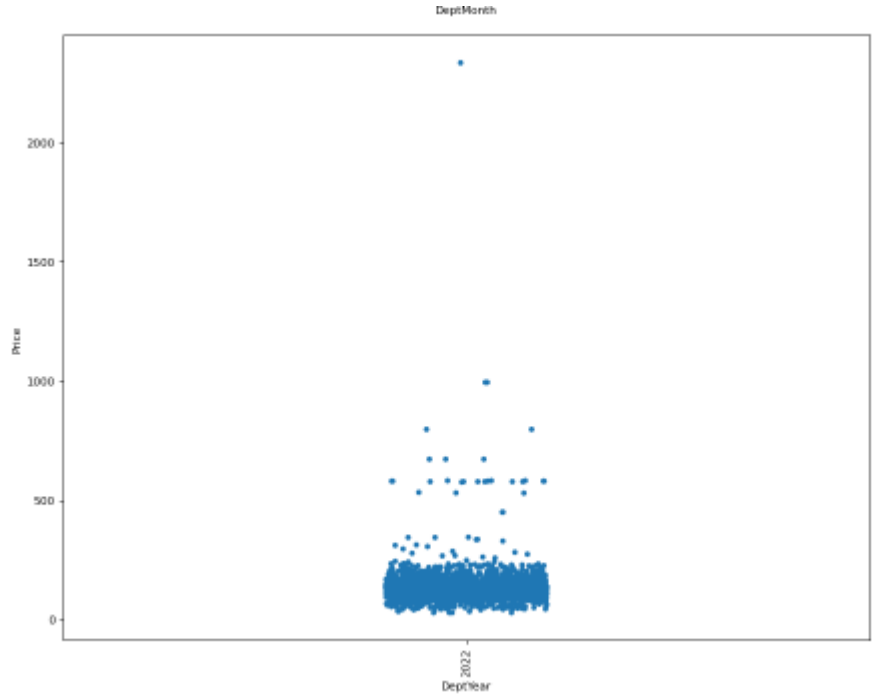

Direct
Price
2000
1500
1000
500
0
BCN Barcelona-El Prat
BVA Beauvais-Tille
ORY Paris-Orly
CDG Charles de Gaulle
CitiesDept

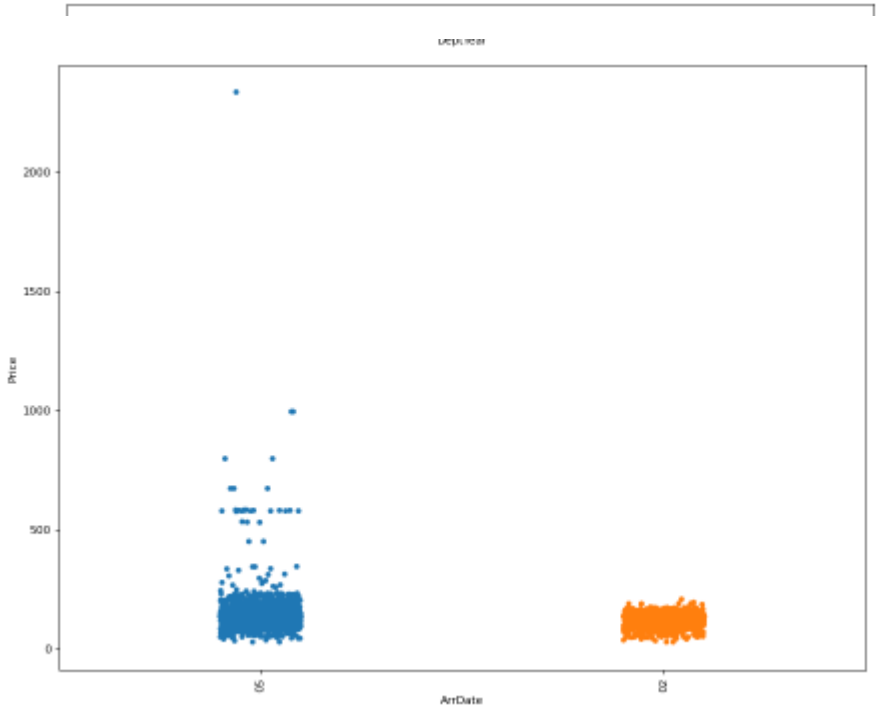

CitiesDept
Price
2000
1500
1000
500
0
ORY Paris-Orly
BCN Barcelona-El Prat
BVA Beauvais-Tillé
CDG Charles de Gaulle
Cities_Arriv

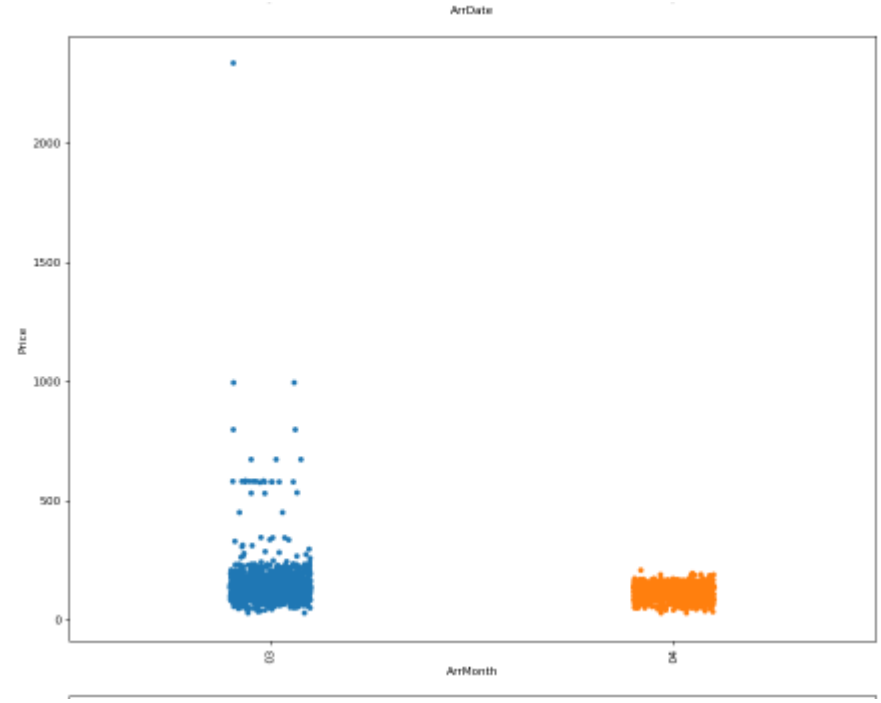

Cities_Arriv
Price
2000
1500
1000
500
0
Price

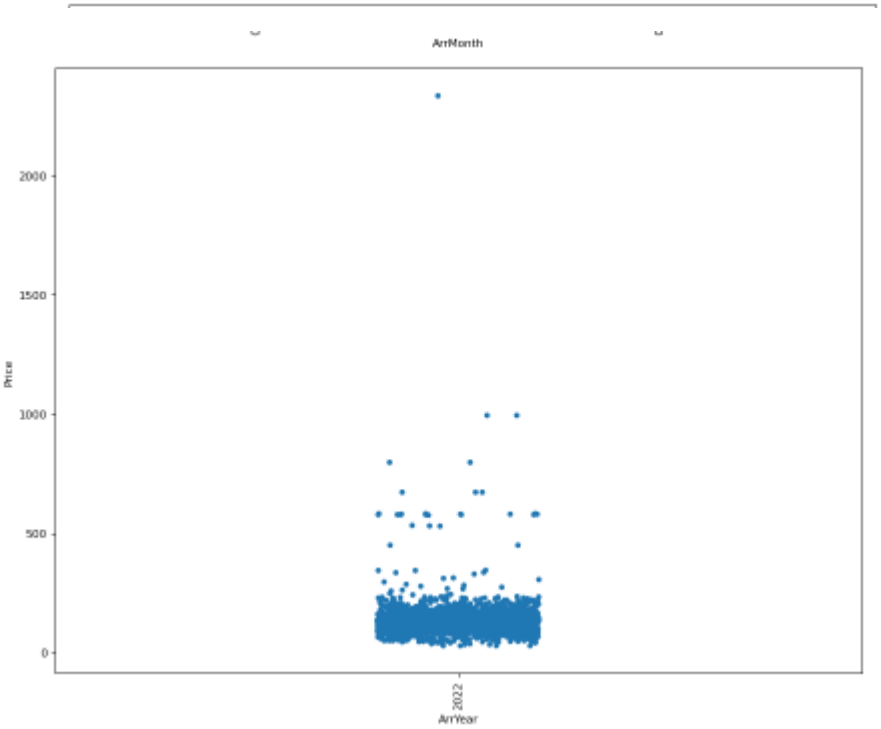

Price
2000
1500
1000
500
0
Price
26
DeptDate

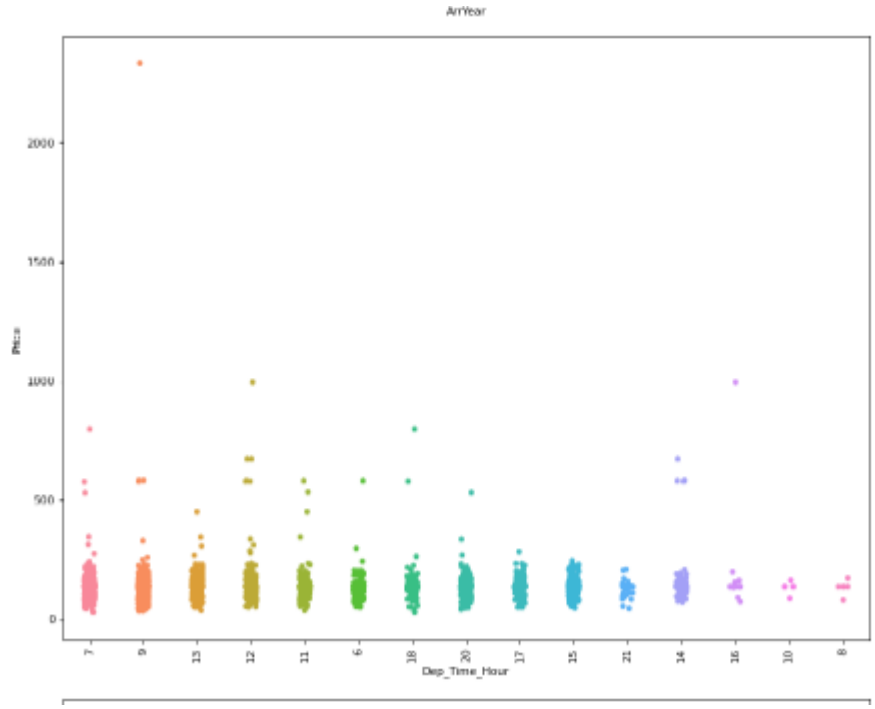

DeptDate
2000
1500
1000
500
0
Price
02
03
DeptMonth

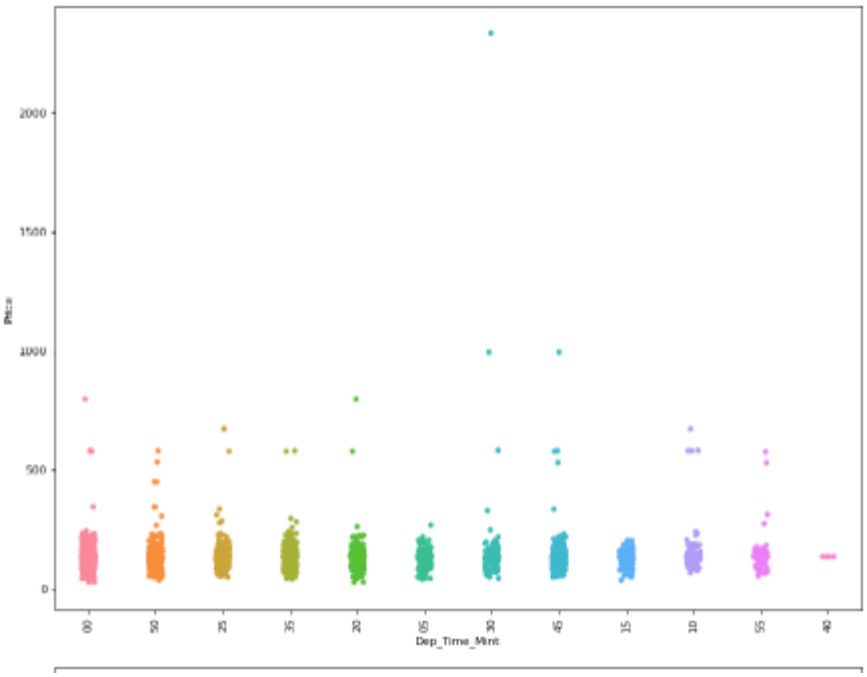

DeptMonth
2000
1500
1000
500
0
Price
2022
DeptYear

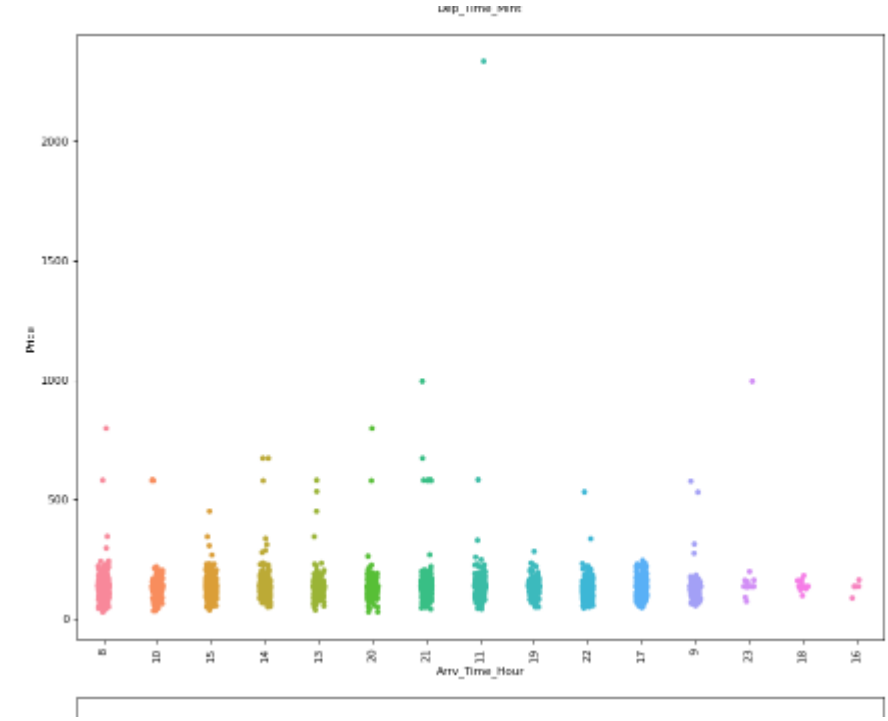

2000
1500
1000
500
0
Price
05
03
ArrDate

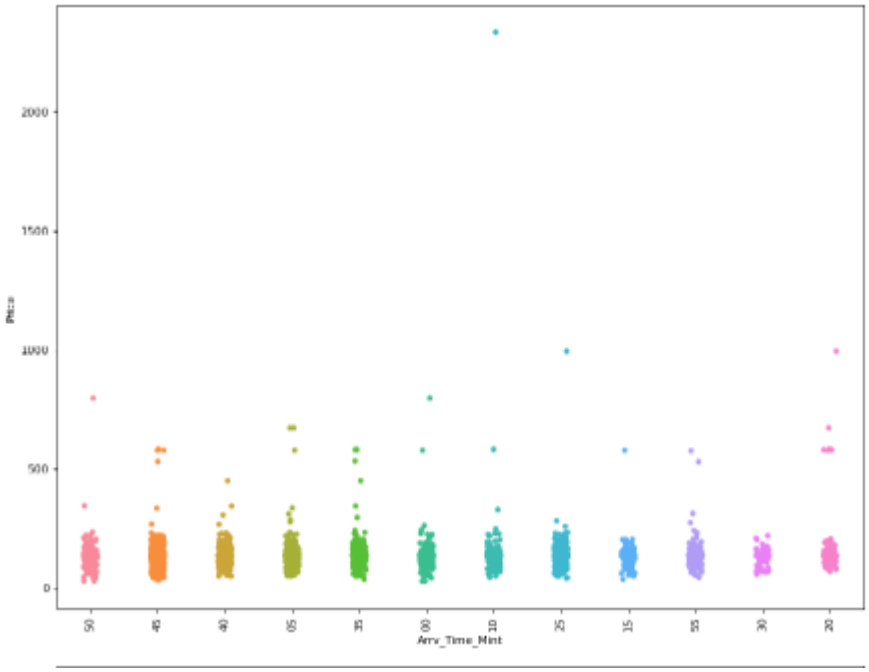

ArrDate
2000
1500
1000
500
0
Price
03
06
ArrMonth

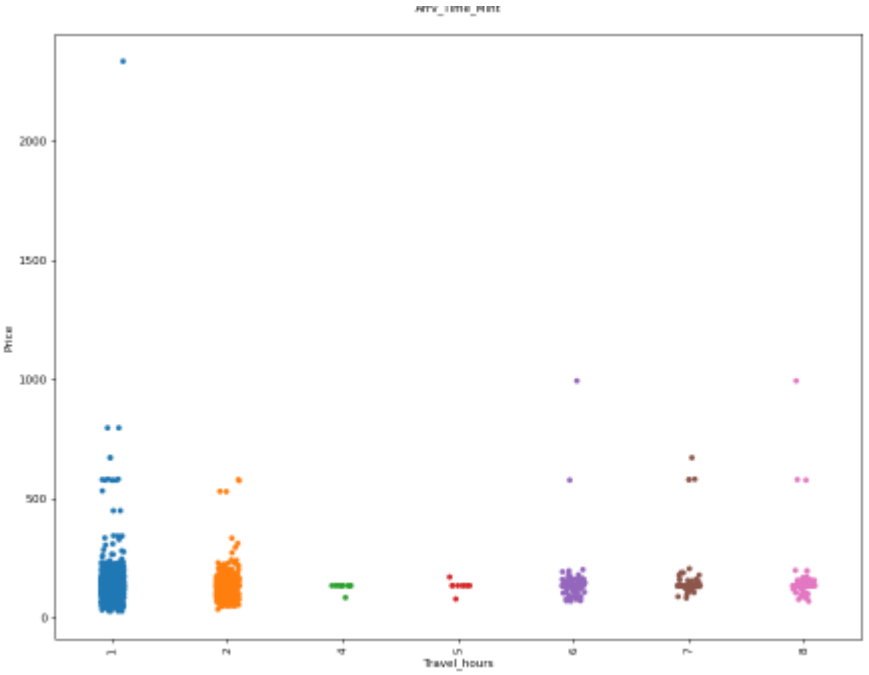

ArrMonth
2000
1500
1000
500
0
Price
2022
ArrYear

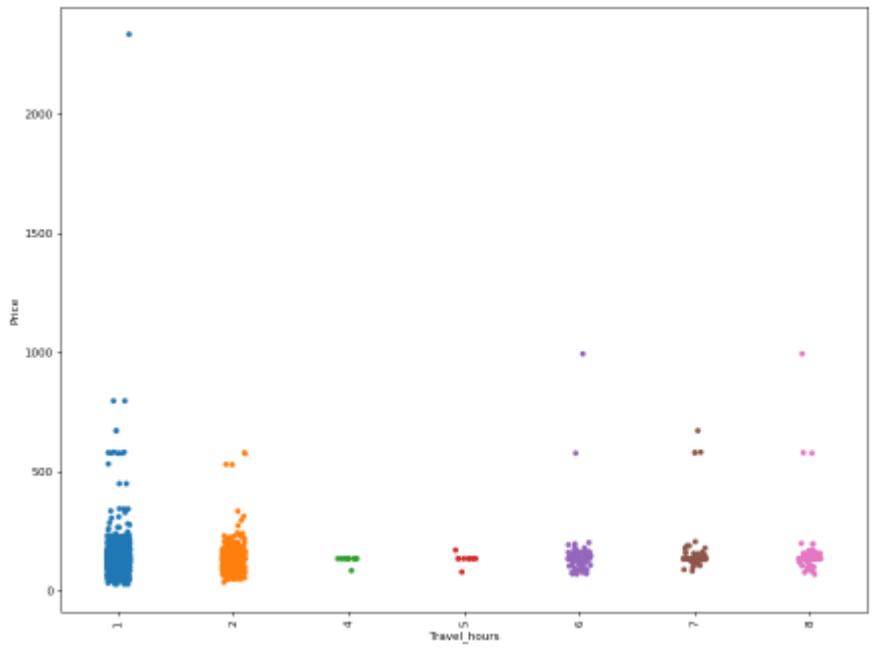

ArrYear
Price
2000
1500
1000
500
0
7
9
13
12
11
6
18
20
17
15
21
14
16
10
8
Dep_Time_Hour

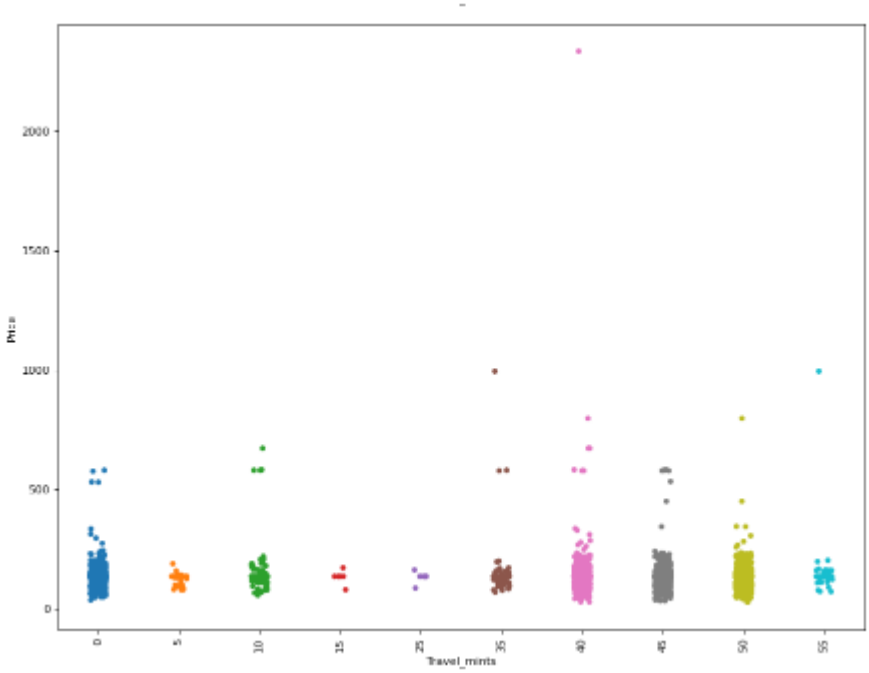

Price
2000
1500
1000
500
0
00
50
25
35
20
05
30
45
15
10
55
40
Dep_Time_Mint


Dep_Time_Hour
2000
1500
1000
500
0
Price
8
10
15
14
13
20
21
11
19
22
17
9
23
18
16
Arrv_Time_Hour


2000
1500
1000
500
0
Price
50
45
40
05
35
00
10
25
15
55
30
20
Arrv_Time_Mint


2000
1500
1000
500
0
Price
Travel_hours


2000
1500
1000
500
0
Price
Travel_hours

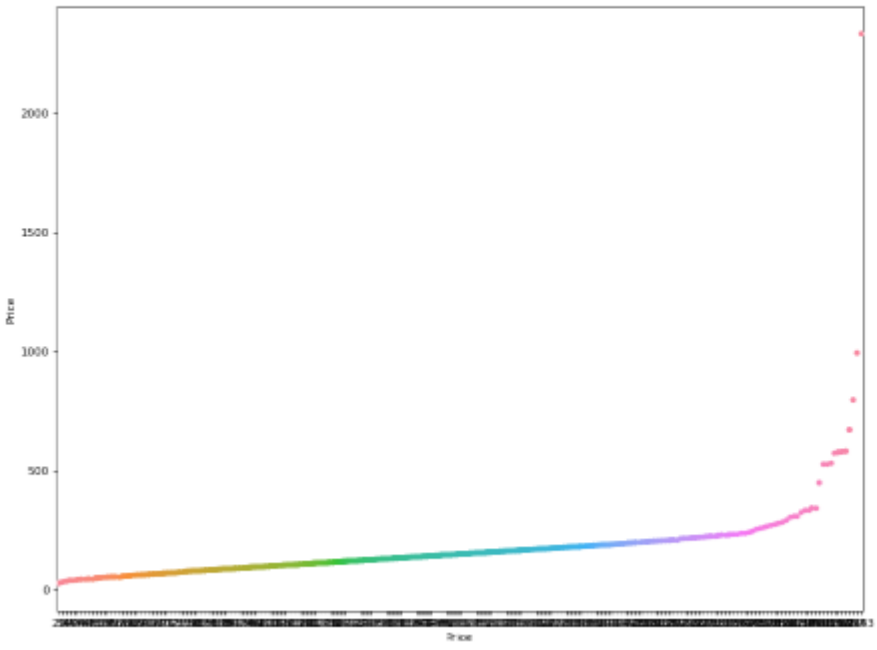


Here above we have the same findings and conclusion we mentioned in the earlier comments.

[86]:
```
#plotting stripplot regarding the reaction of prices with regard Price itself:
plt.figure(figsize=(13,10), dpi= 80)
sns.stripplot(x='Price', y="Price", data=df, jitter=True)
ax.set_xticklabels(df.Price,rotation='vertical')
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/ipykernel_launcher.py:4: UserWarning: FixedFormatter should only be used together with FixedLocator
  after removing the cwd from sys.path.
```

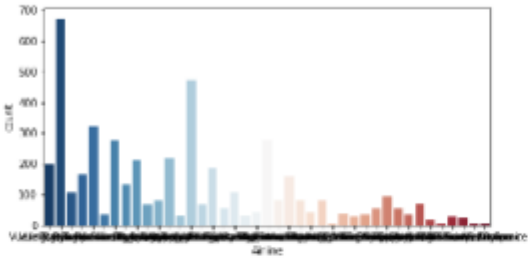
[86]:
```
[Text(0, 0, '29'), Text(1, 0, '34')]
```

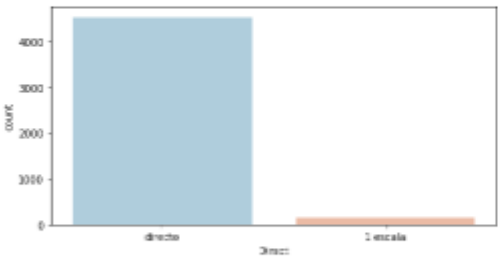


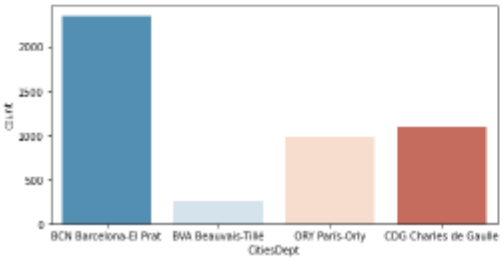
We have Prie increase as the Price increases.

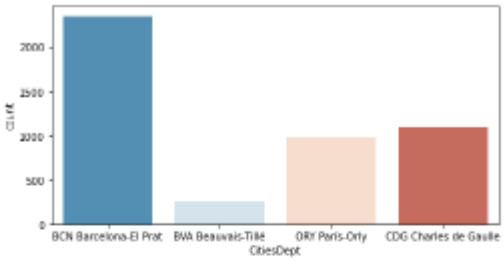
We have Prie increase as the Price increases.

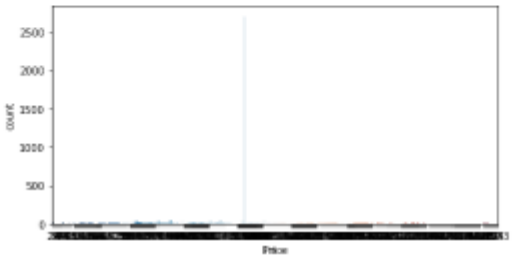
[87]:
```
#countplotting regarding the realtion of prices with all columns except the ones that are list by itself as we have already columns having the same info:

for col in colsnotlist:
    plt.figure(figsize=(8,4))
    sns.countplot(x=col,data=df,palette='RdBu_r')
```

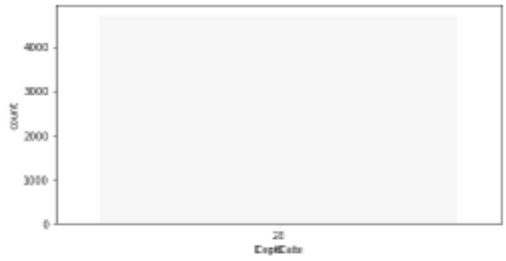


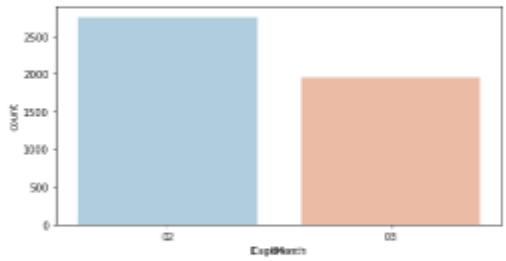




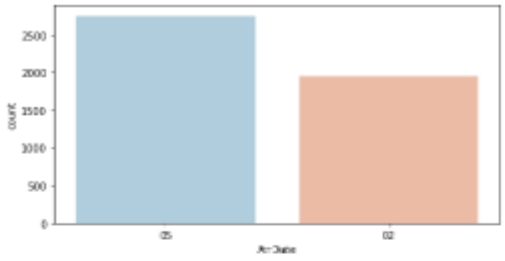


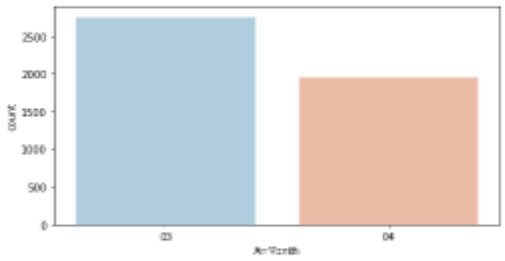



count
4000
3000
2000
1000
0
28

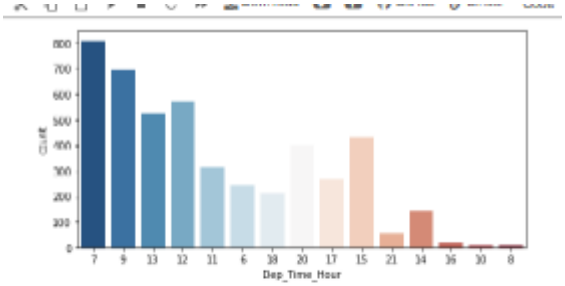

count
2500
2000
1500
1000
500
0
02
05

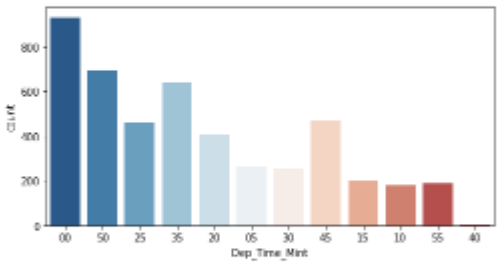

count
4000
3000
2000
1000
0
2022

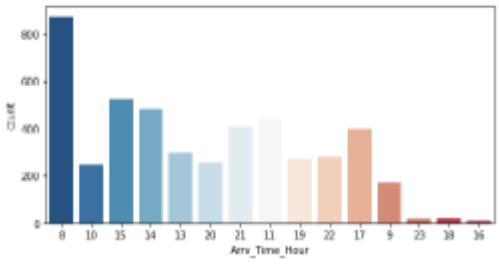

count
2500
2000
1500
1000
500
0
05
02

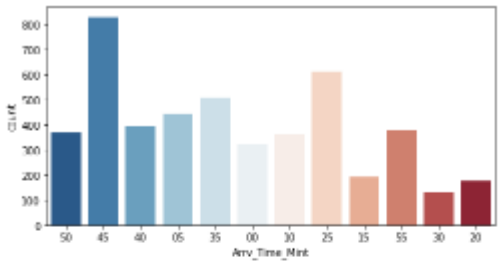

count
2500
2000
1500
1000
500
0
03
04

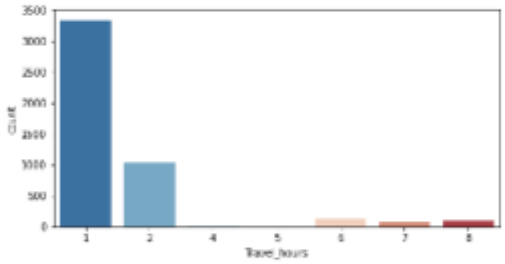

count
4000
3000
2000
1000
0
2022

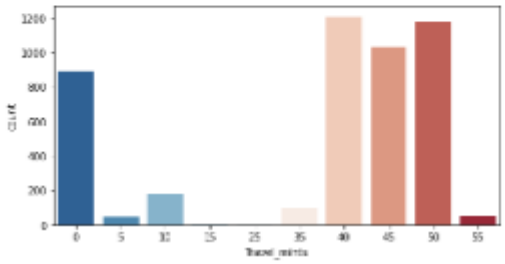












PLEASE BEAR IN MIND THESE CONCLUSION AND FINDINS ARE DEPENDENT ON THE DATA WE GOT, WHICH MEANS IT DEPENDSON ON THE SOURCE AND DESTINATION OF FLIGHT. AND ALSO ON DEPARTURE AND ARRIVAL DATE SELECTED FOR THE DATAFRAME. HENCE, ALL

OUR CONCLUSIONS ARE DEPENDENT ON THESE 4 SELECTIONS AND CAN ONLY BE APPLICALE ON FLIGHTS GOING BETWEEN BCN AND PARIS.

Here we have and notice the same findings as previously mentioned.

In addition, we have the following highlights:

- Vueling and combo of Ryanair and TransaviaFrance has the highest number of flights offer.

- Most of the flights are direct flights.

- The city that depart the most is Barcelona.

- The city that has most of the arrivals is also Barcleona

- We have one unique Departure Date which is 26th. The same happens with Year, unique value of 2022.

- The month we have are February and March. February has more flights than March.

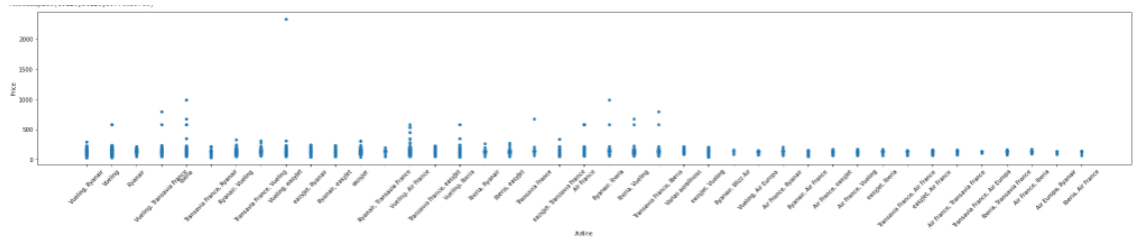
- As we said, when comparing ArrDate and ArrMonth, we see a tendency to have more flights near to current date than far away.

- When comparing Dep Time Hour and Mint, we see hours like 7am,9am,13h and 12h have more probability to have flights available than the remainign departures timing. And regarding the Dep Time Mint, we have that most of the flights departs at minutes 00, 50, 35, and 25. And the timing where less flights are offered tend to be at 16, 10, and 8am for Paris-Barcelona.

- And regarding the arrival timings, most of the flight offers tend to be most probable at 8am and less probable to be at 23, 18, or even 16h. And when comparing the minutes, we have most of the flights to be at 45min 25min, 35min and 05minutes.

- When comparing Travel_hours, we see it is most probably to be between 1h and 2h. There is way less flights which travel duration is more than 2h.

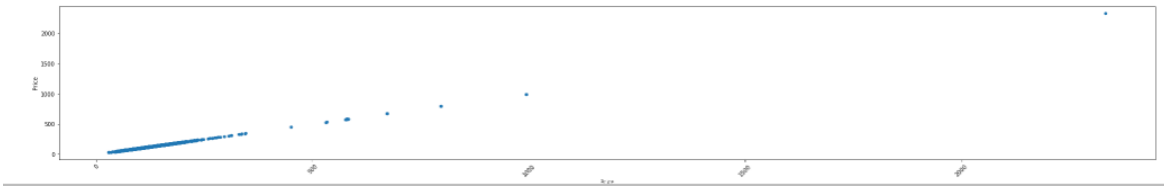
- When comparing Travel_minutes, we see it is most probably to be between 40min and 50min. There is way less flights which travel duration minutes ends in 15minutes or 25 minutes.

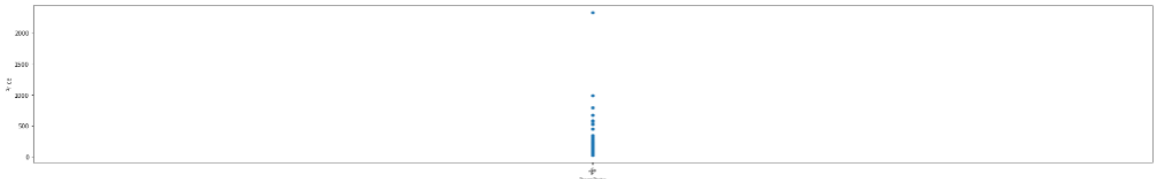

Price
Airline

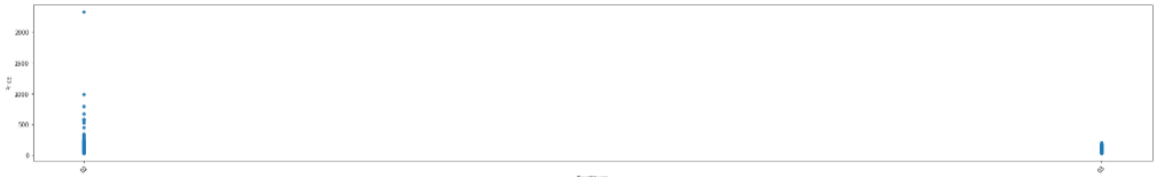

Price
Direct

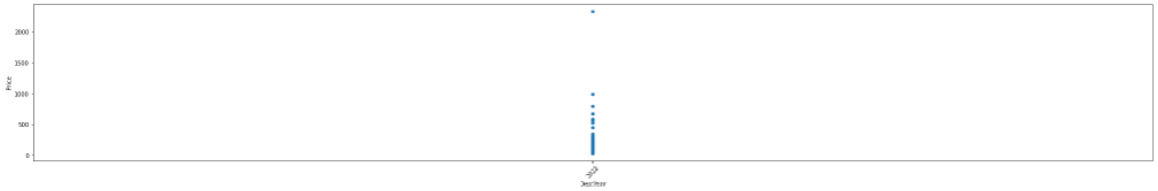

Price
DépartOrigine

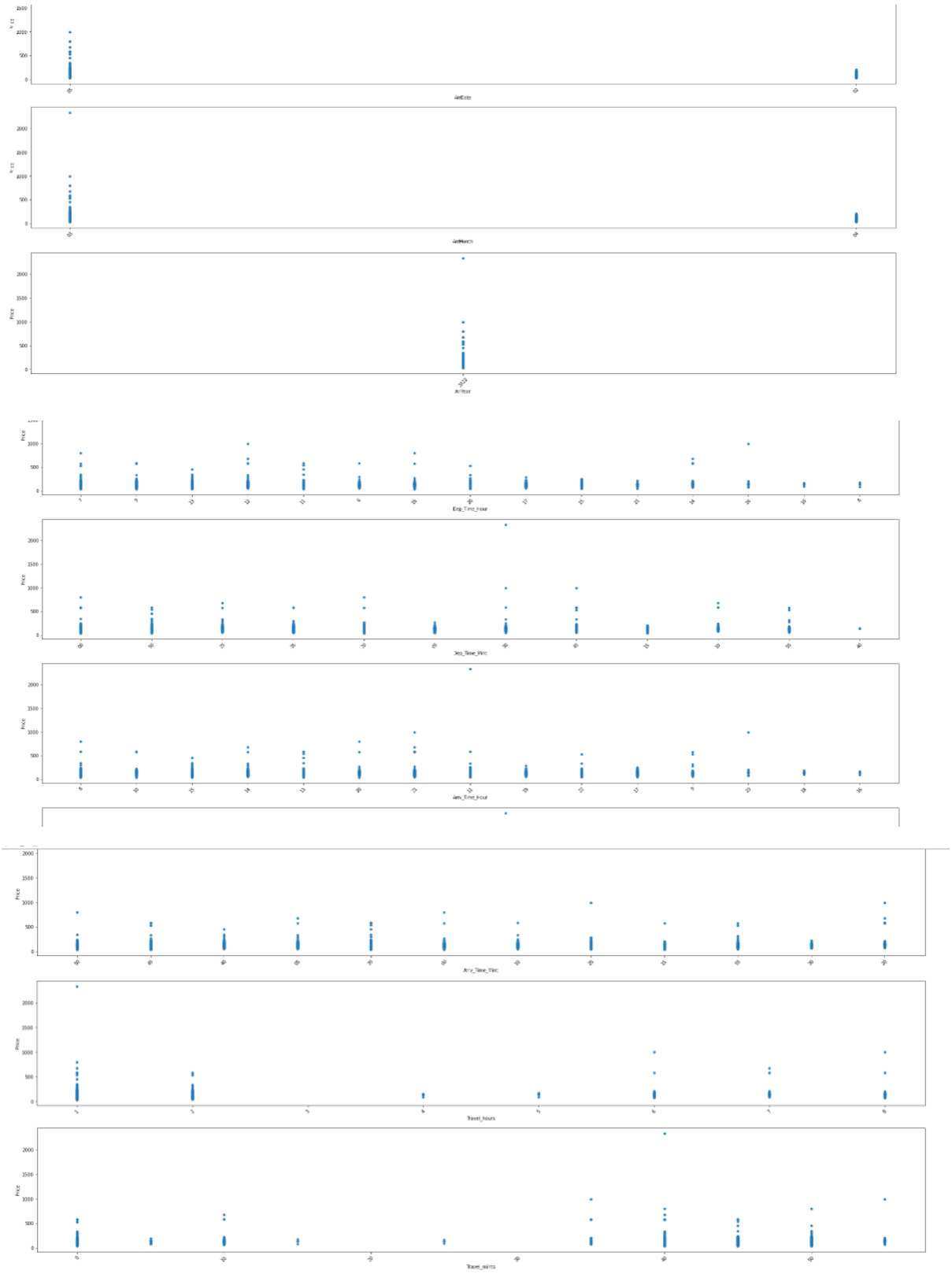

Price
Départ_Arrivée


Price


Price
Jour_Départ


Price

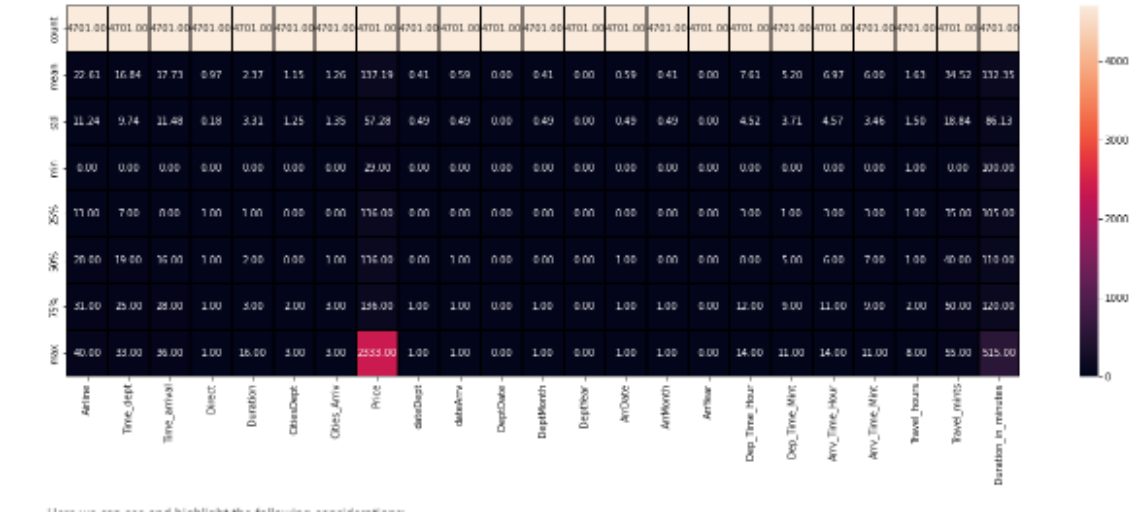

Price
Jour_Retour



In the above plot we have see the same conclusions as previously mentioned.

```
[94]: #statistical description of all numerical columns:
      import matplotlib.pyplot as plt
      plt.figure(figsize=(22,7))
      sns.heatmap(df.describe(),annot=True,linewidth=0.1,linecolor="black",fmt="0.2f")
```

```
[94]: <AxesSubplot:>
```



Here we can see and highlight the following considerations:

- we have no missing values at this stage as the first row mentions total rows of dataframe we have in all our columns.

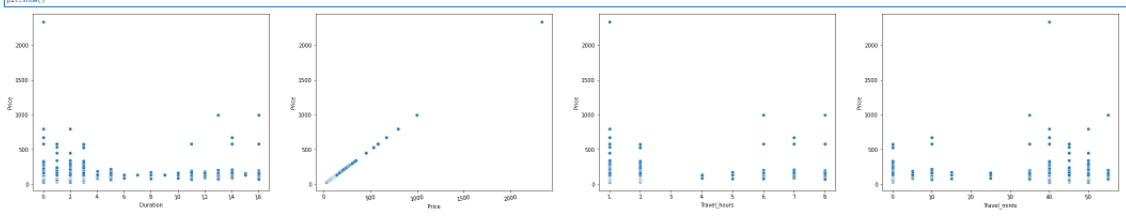
- we have columns like Time_arrival, Duration, CitiesDept, Cities_Arrv,Price, dateDept, DeptMonth, ArrMonth, Dept_Time_Mint, Arrv_Time_Hour, Travel_hours and Duration_in_minutes which have higher mean than the median which hence means that we have right side skewed distribution in these features.

- And DeptDate and DeptYear are non skewed data as there is only 1 unique value. The same happens with Arrival Year.

- The remaining features have left side skewed distribution, which means mean is smaller than the median.

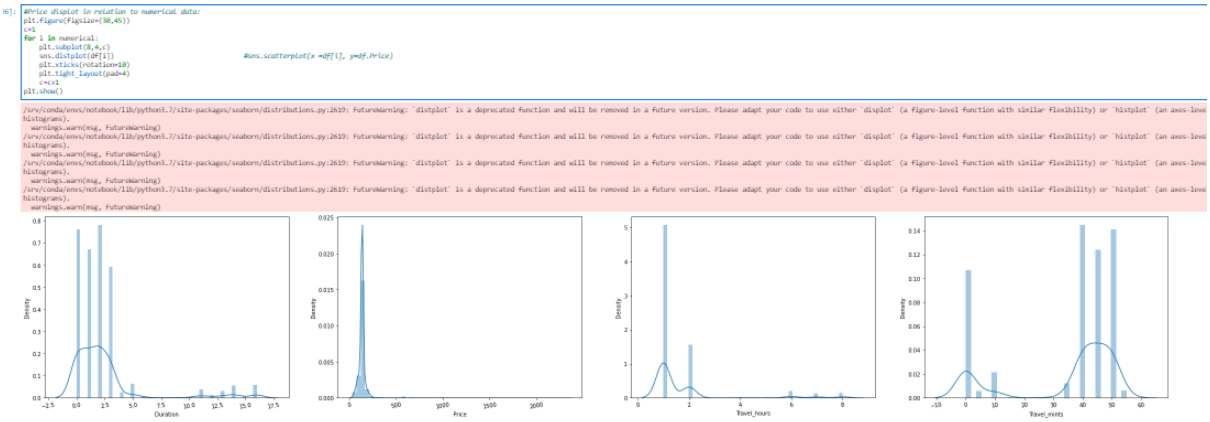
- We have that all columns except Direct, Cities_Ariv, dateDept, dateArriv, Depttdate, DepatMonth, DeptYear, ArrDate, ArrMonth and ArrYear have outliers in them as the max is higher than the 75% percentile.

```
[95]: #Price scatterplot in relation to numerical data:
      c=1
      plt.figure(figsize=(30,45))
      for i in numerical:
          plt.subplot(8,4,c)
          sns.scatterplot(x =df[i], y=df.Price)
          plt.xticks(rotation=10)
          plt.tight_layout(pad=4)
          c=c+1
      plt.show()
```



Here above we can find that the only linear relationship we find is the one betwen price and Price dues to its 1x1 relation.

Regarding the remaining numerical features, we see no linear relation between price the the corresponding features.

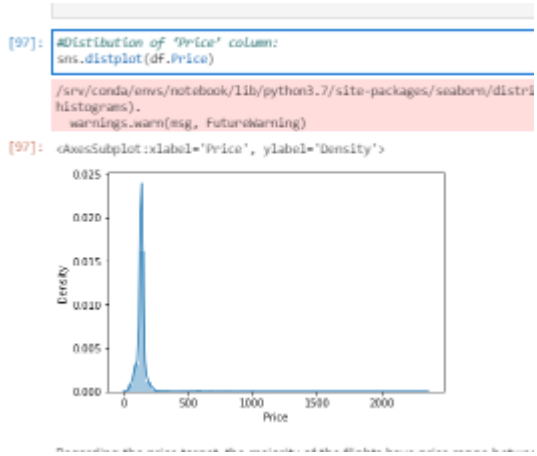


We see that Duration seems to be between 0 to 2:30h.

Price seems to be between 0 and 250 euros per person.

Travel hours are mostly on the whole hour and happens to be mostly betwween 0 to 2 hours.

Travel minutes:The only thing that stands out is the travel minutes between 40 and 50 minutes are the most frequent ones. And in third place, we have flights whose travel minuts is 00.

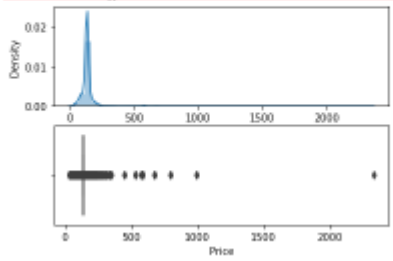


Regarding the price target, the majority of the flights have price range between around 0-250 euros and then the flights decreses substantially after the price range over 250. Flights having prices greater than 20k are quite less. Price range is skewed towards right. As we said earlier in the analisis, we have right skewness in the target price.

We will now compare several independent features with ‘Price’ column to check their impact:

[99]:
```
#Let's plot 'Price':
plot(df,'Price')
```

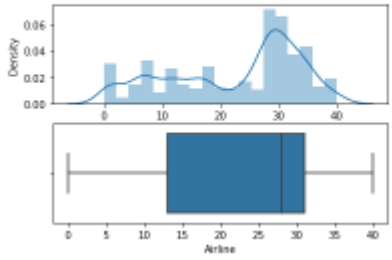
```
/srv/conda/envs/notebook/lib/python3.7/site-packages/seaborn/distributi
histograms).
  warnings.warn(msg, FutureWarning)
/srv/conda/envs/notebook/lib/python3.7/site-packages/seaborn/_decorator
  FutureWarning
```



If we check density, as we said, we have range of 0 and 260 euros. And exepceptionally we have price of 2k. So, the distribution has right skewness.

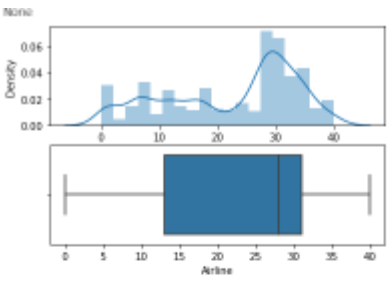
And then, if we check boxplot, we see a median on around 9k which means outliers taking place between 100 euros and 120 euros are impacting way more than the other cases.

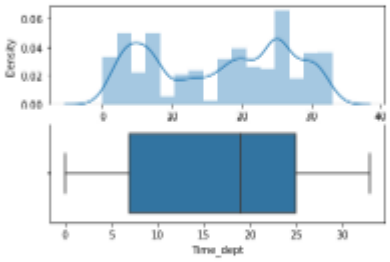
```
/srv/conda/envs/notebook/lib/python3.7/site-packages/seaborn/distrib
histograms).
  warnings.warn(msg, FutureWarning)
/srv/conda/envs/notebook/lib/python3.7/site-packages/seaborn/_decora
  FutureWarning
```

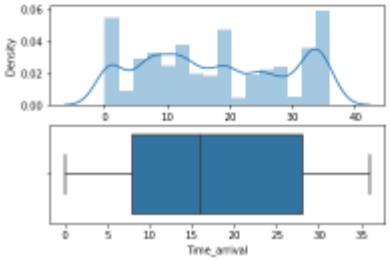


Here we can see few airlines like Vueling and Ryanair have more number of flights offered than others. Also few combos like Vueling and Air France.

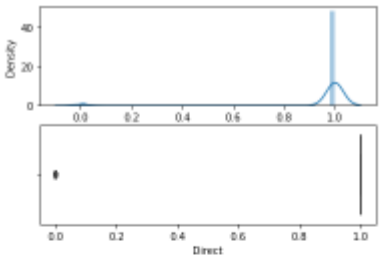
None

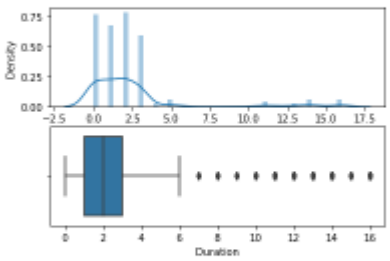

Density
Airline

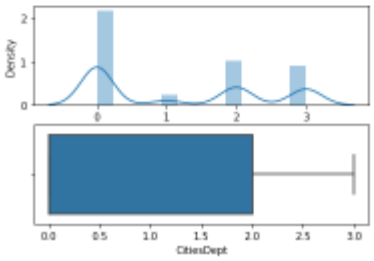

Density
Time_dept

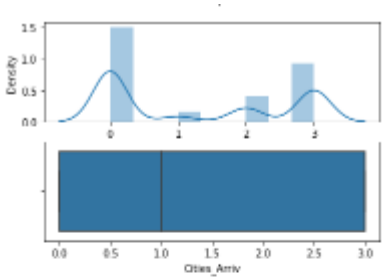

Density
Time_arrival

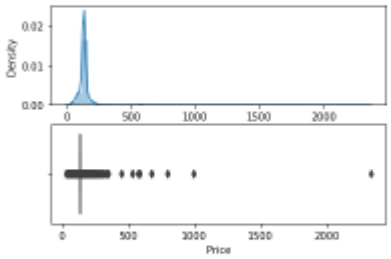
-

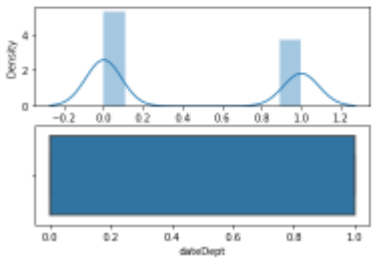

Density
Direct

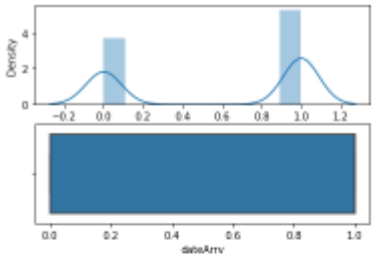

Density
Duration

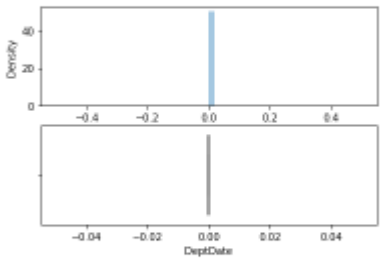

Density
CitiesDept

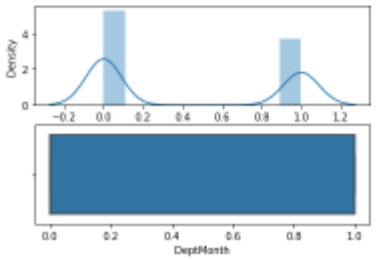

Density
Cities_Arriv

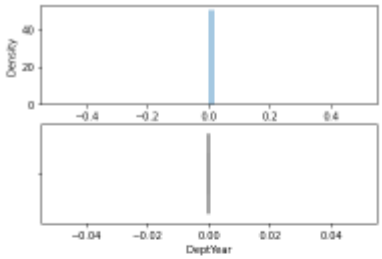

Density
Price

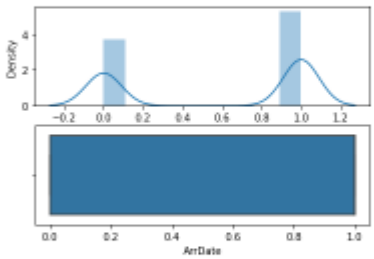

Density
dateDept

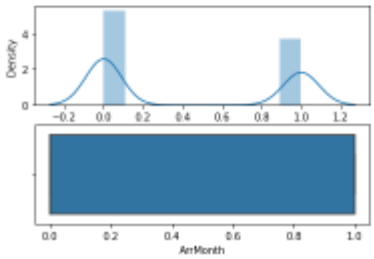

Density
dateArriv

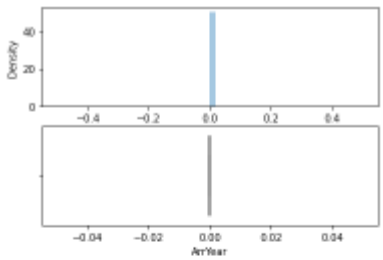

Density
DeptDate

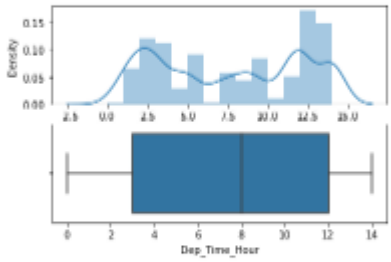

Density
DeptMonth

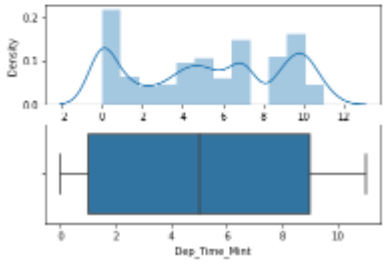

Density
DeptYear

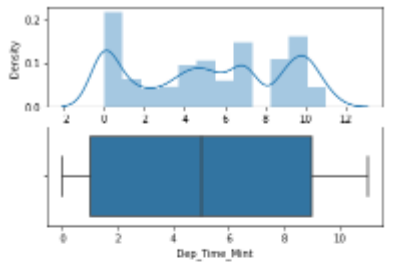

Density
ArrDate

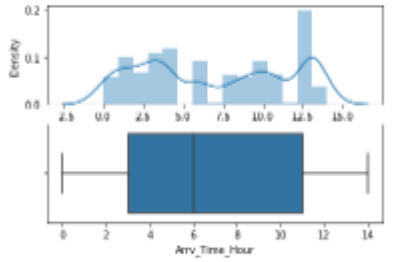

Density
ArrMonth

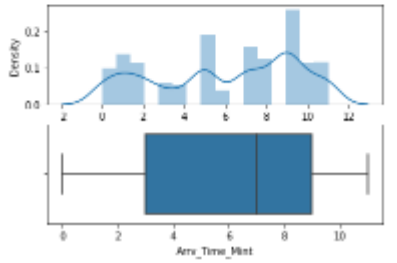

Density
ArrYear


Density
Dep_Time_Hour

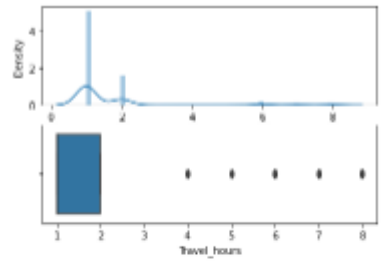

Density
Dep_Time_Mint

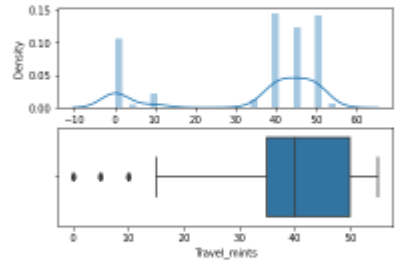


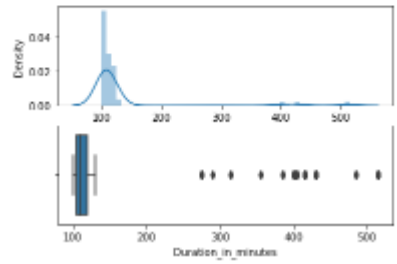




Arrv_Time_Mint







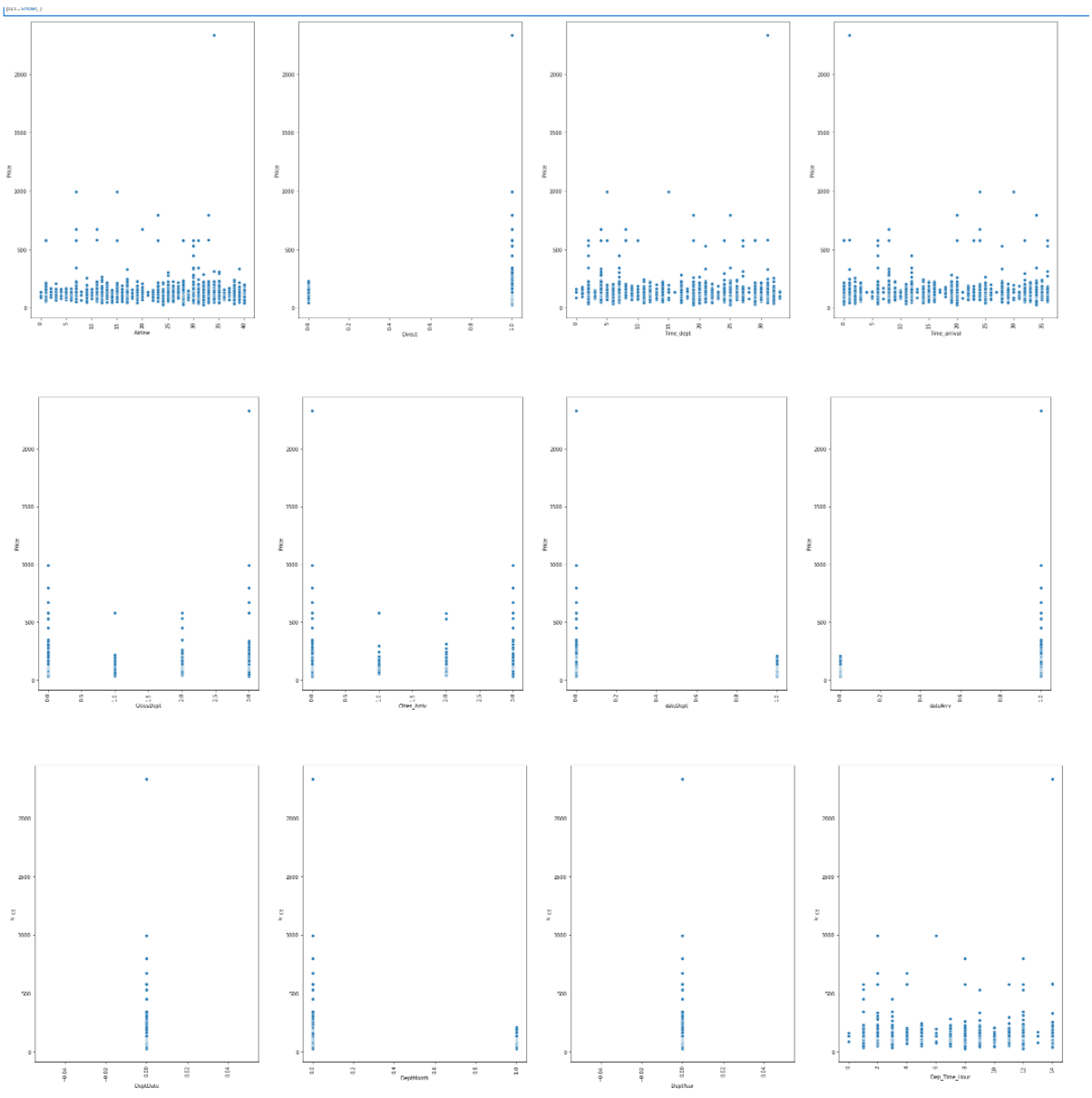
As we previously said, here we can easily notice that some columns have right side skewed data distributions and other left sided:

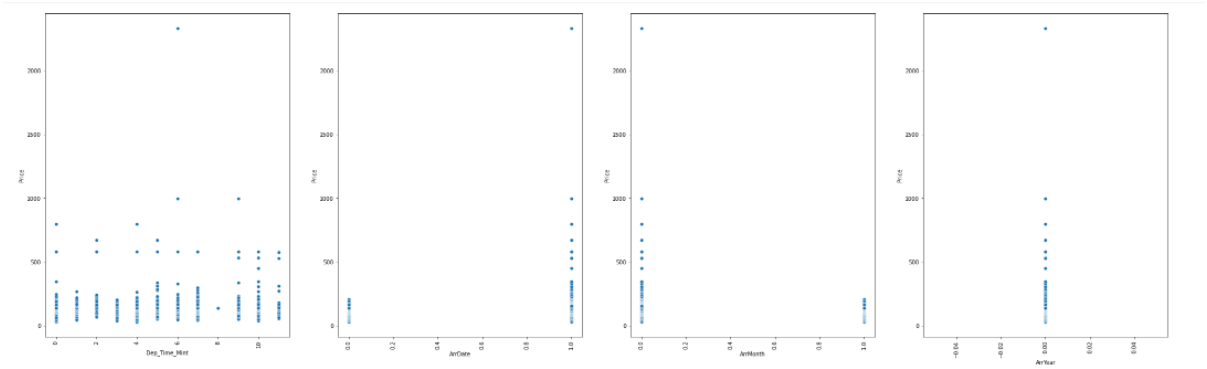
We have columns like Time_arrival, Duration, CitiesDept, Cities_Arrv,Price, dateDept, DeptMonth, ArrMonth, Dept_Time_Mint, Arrv_Time_Hour, Travel_hours and Duration_in_minutes which have right side skewed distribution in these features.

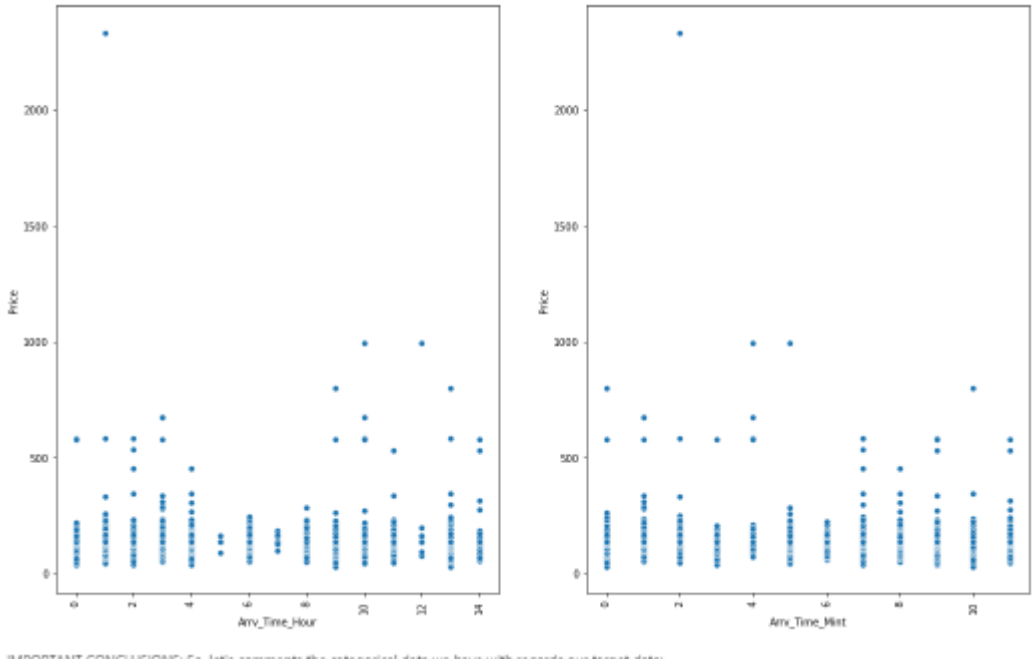
- And DeptDate and DeptYear are non skewed data as there is only 1 unique value. The same happens with Arrival Year.

- The remaining features have left side skewed distribution,as we easily can see in the above distribution plots.

Here in the boxplot, we can see we have that all columns except Direct, Cities_Ariv, dateDept, dateArriv, Depttdate, DepatMonth, DeptYear, ArrDate, ArrMonth and ArrYear have outliers in them as the max is higher than the 75% percentile.







IMPORTANT CONCLUSIONS: So, let's comments the categorical data we have with regards our target data:

- we have different type of occurance of flights of differents airlines, hence the prices varies depending if the choosed flight is a combo or not.

- Departure flighted date is stable

- we have more direct flights than flights with a stop. Here we can see the direct flight has more variance in the flight price than the flight with a stop. Maybe the case that we have unbalance data if we check as per direct or not direct flights.

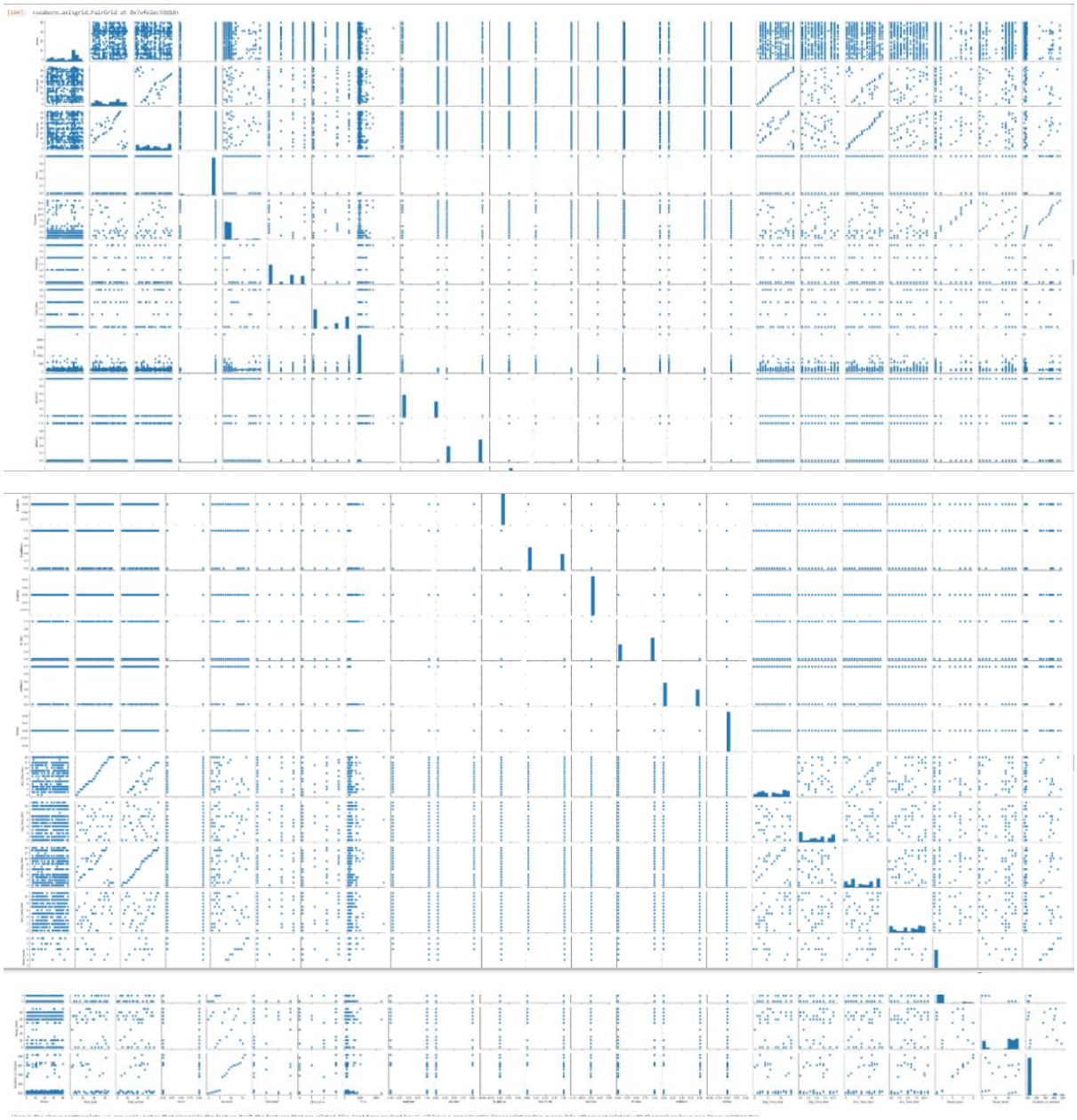
- When checking the price by departure time and by arrival time, we see we have a huge variance in the price depending on the departure and arrival time.

- When checking flight prices by the Cities of departure or Arrival cities, we see the cities airport which gets most of the flights has a greater variance of prices compared with other airports.

- As we said, when checking the departure date and arrival date, we may have found that nearest flights has more offer than flights that are planned to be taken later. Which also

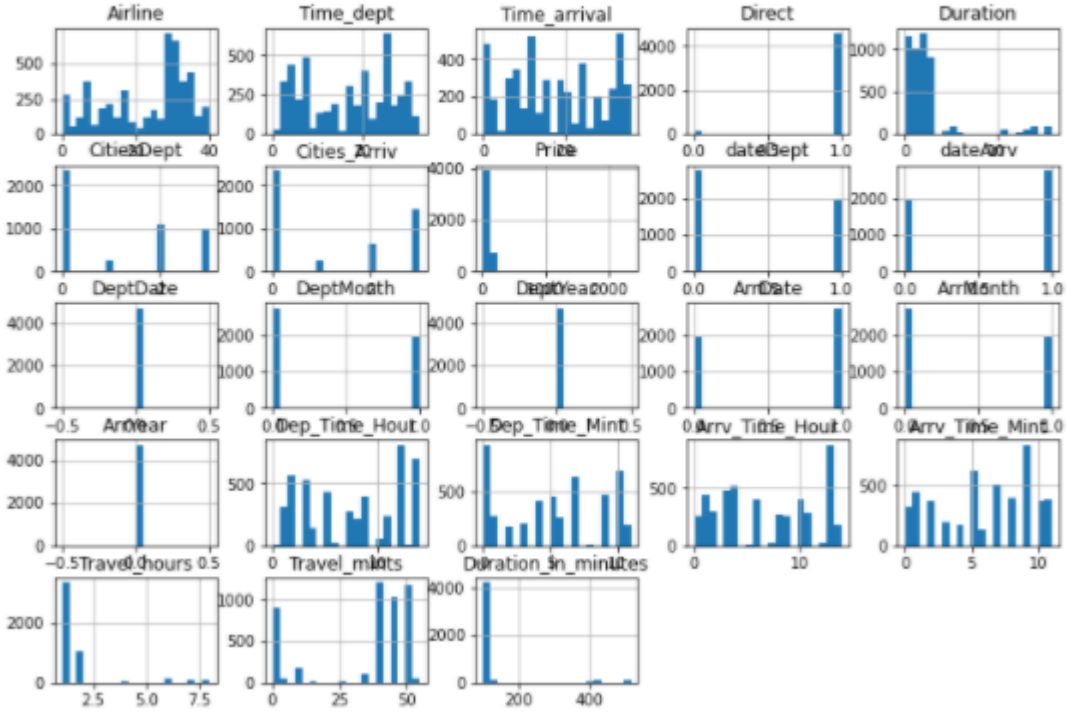
means at the same time, that the prices will have greater variance for flights that will be flown earlier than flights that will depart or arrive few days or months later.

- The airline flight price gets lower or higher price depending on the time, hour and minute it is taken from the airport. :)

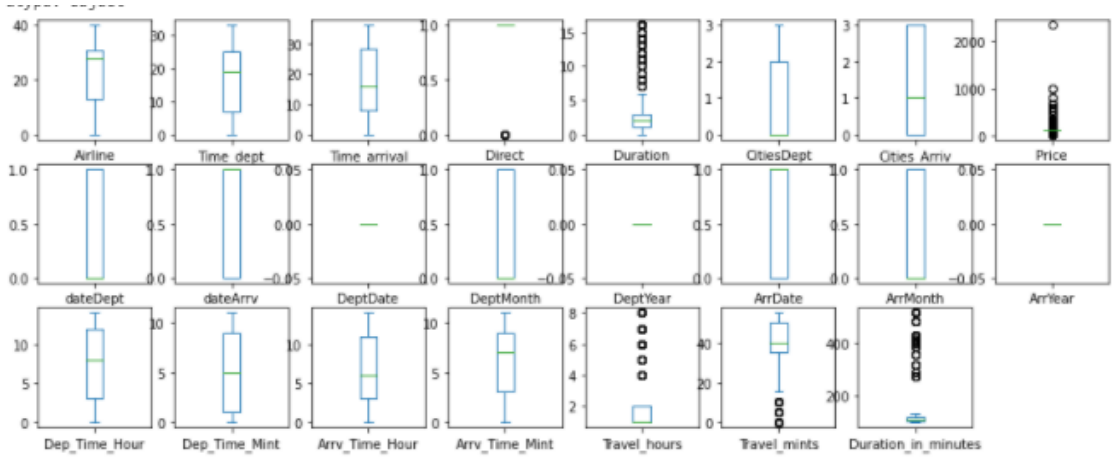


Here in the above scatter plots, we can easily notice that alongside the feature itself, the features that are related (like dept time or dept hour) will have a considerable linear relationship meanwhile others not related with themselves have non linear relationship.

```
[105]: #Distributions of the variables/features.
df.hist(figsize=(12,8),bins=20)
plt.show()
```



In the above barplots, we can easily see we have skewed data in almost all our features including Price.



As we have already seen several times in our previous boxplots, we have many outliers in several features columns such as:

- Direct, Duration, Price, Travel_hours, Travel_minutes and Duration in minutes.

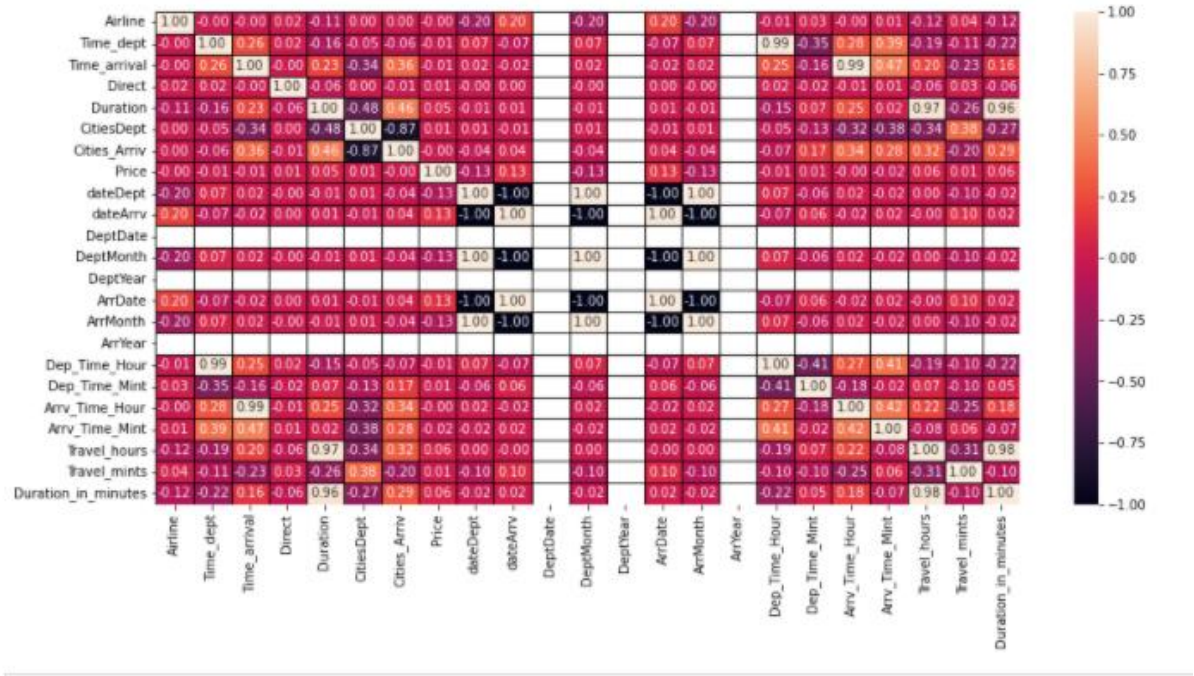
But from here we can conclude the following as we have the whole picture of all the outliers: Depends on the proportion of the outliers with regards the whole data, we can or can not remove. And also we need to mention is the following:

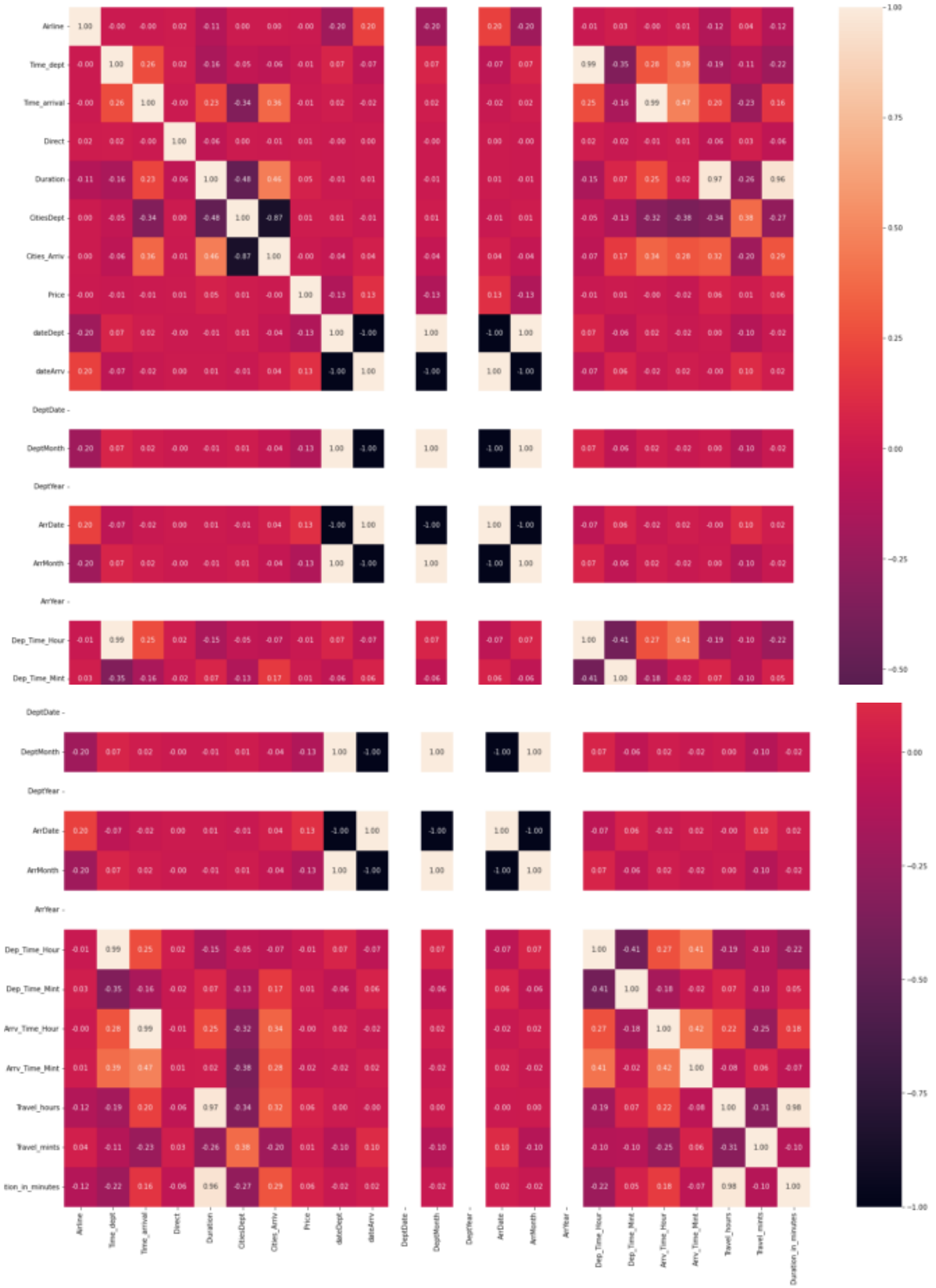
- We will not remove outliers like from total stops since price is impacted by number of stops.
- Happens the same high hours of travel, it impacts more as long as the hours get increased.

This column Year has only 2022 as a value and can be dropped it.

```
#Correlation using heatmap:
import matplotlib.pyplot as plt
plt.figure(figsize=(15,7))
sns.heatmap(df.corr(),annot=True,linewidths=0.5,linecolor="black",fmt='.2f')
```

```
<AxesSubplot:>
```





Regarding the target Price, we see the features that are highly correlated with it are the Departure and Arrival Date.

We also see that:

- Departue city and Ariival city are highly correlated with Duration and Arriva Time.

- We also see the departure City and arrival city are also very highly correllated.

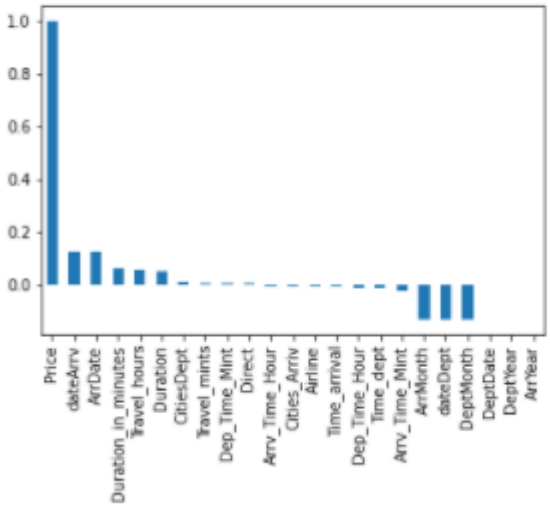
- We see departure time is almost fully correlated with departure time hour. And same happens with Arrival case, we see arrival time is almost fully correlated with Arrival time hour.

- And as it is expected, travel hours is almost fully correlated with duration in minutes and the same time these are almost fully correlated with duration.

Let's now check plot of the correlation of the features with regards the target feature in the following code:

```
[61]: #correlation of independent features with regards the target "Price"
correlations =df.corr()['Price'].sort_values(ascending=False)
correlations.plot(kind='bar')
```

```
[61]: <AxesSubplot:>
```

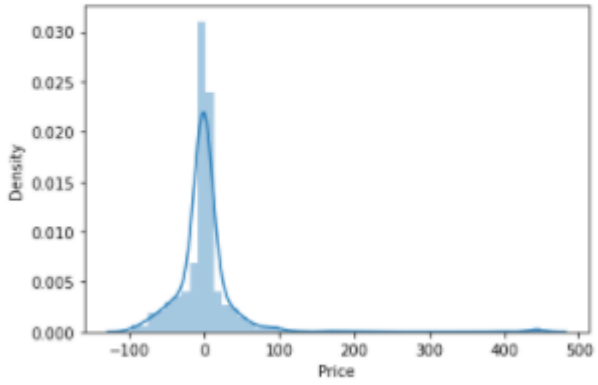


As we can see, departure date and Arrival date have the most impact on our Price target feature. And in third place, we have Duration in minutes, Travel hours and Duration which has considerable impact on our target Price.

After the above analysis, we drop the non-required columns, which we feel have no impact on prices of flights. These columns include all the splits from the date and time of the flights.

```
prediction=res3.predict(X_test)
sns.distplot(y_test-prediction)
```

```
<AxesSubplot:xlabel='Price', ylabel='Density'>
```



And finally the above type plot represents residuals which is the difference between the expected values and the actual values.

- ## Interpretation of the Results

  Give a summary of what results were interpreted from the visualizations, preprocessing and modelling.

  Regarding Visualizations, we highlight the following items:

  PLEASE BEAR IN MIND THESE CONCLUSION AND FINDINDS ARE DEPENDENT ON THE DATA WE GOT, WHICH MEANS IT DEPENDS ON ON THE SOURCE AND DESTINATION OF FLIGHT. AND ALSO ON DEPARTURE AND ARRIVAL DATE SELECTED FOR THE DATAFRAME. HENCE, ALL OUR CONCLUSIONS ARE DEPENDENT ON THESE 4 SELECTIONS AND CAN ONLY BE APPLICALE ON FLIGHTS GOING BETWEEN BCN AND PARIS.

  As such, these can be helpful to understand other flights patterns and check overall price changes insight but having in mind the dates and the cities, since the price change from one cities to another changes depending on a city and also for which period we want to travel.

Regarding flights having similarities to Paris-Bcn characteristics, we can mention the following finfinds:

We can see all airlines have similar price median but some airlines have few cases of higher flight prices compared to others. We can come up with a conclusion all airlines had almost similar median with minimal fluctuations.

- We could see Vueling alone has the highest number of flights. Then in second position we have combo of Ryanair and Transavia France which comes in 2nd place and get the most flights after Vueling.
- Then, in third place, we have combo of Vueling and Transavia France which gets most flights in third place.
- When checking the flights counts by Time of departure, we see 7am has the highest number of seats available. Then, after that, in second place comes flight that departs at 12:25h. And in 3rd place, comes the flight that departs at 13:50.
- When comparing the lowest number of seats available, we see that we have 16:40,10:05,8:35h.
- When checking the flights counts by Arrival Time, we see 15:40h has the highest number of seats available. Then, after that, in second place comes flight that arrives at 08:50h. And in 3rd place, comes the flight that departs at 14:05.
- When comparing the lowest number of seats available, we see that we have 22:35,16:30,13:50 and 11:50.
- As these flights are between Barcelona and Paris, it is obvious that most of the flights would be direct flights. There is just one exception which has 1 stay in between.
- When checking flights by duration, we can see flights with duration etween 1:45h and 2h are the most used and available in the website.
- When checking flights by Departure Cities, we see Barcelona gets the 1st position as the city from which the flights gets mostly departed as Barcelona is the main Airport in Spain. And in sencond place, from France we have 3 options as Departure Airports but the french airport which gets the position when comparing the flight departure is Charles de Gaulle which is the main airport of Paris.

- When checking flights by Arrival Cities, again we see Barcelona gets the 1st position as the city to which flights mostly arrives as Barcelona is the main Airport in Spain. And in sencond place, from France we have 3 options as Departure Airports but the french airport which gets the position when comparing the flight departure is Paris-Orly which is the second main airport of Paris.

As we mentioned previously:

Regarding the departure time, we have few timings that may have higher prices than the mean and which are flights departing at 7am, 9am, 13:50pm, 12:25 pm, 11:50am, 18:20, 9:30am, 20:45, 7:55am, 14:10, 12:45pm, 14:35 and 16:30h.

Similar situation happens with Arrival Time, we have Price around the mean and the median but we have also Arrival Time where we have a bit higher prices such as in timings like 8:50, 10:45 (one case), 15:40, 14:05, 13:35,8:35 (one case), 20h (2 cases), 21:45, 11:10, 22:45 (2 cases), 9:55, 21:20, 21:15 (one ase) and 23:25 (one case).

When comparing Duration, it can be studied and analysed the same as the departure and arrival time. Almost al the prices are between 0 and 500 euros except some cases where we have a bit higher prices such as duration of 1h 50min, 1h 45m, 1h 40m (we have 1 outlier here), 2h, 7h 10m,, 8h 35m, 6h 40m and 6h 55m.

Regarding the dates, there is no big change in the Price when checking the flights for between dates 26 march and 2 April as it is just prior to Holy Wweek and the prices are down and close to the mean price while the prices between dates 26 february and 5 March tend to vary more than the other case.

We can see the following highlights:

- The airline with most flights offers is Vueling which top the list with 672 flights. Then, in second place, we have combo Vueling and Air France which offers 473 flights. Then, in third place, we have Iberia with 320 flights. And the flights combo Iberia and Air France or even Air Europa with

combination Ryanair or even Transavia and Air Europa are the combos which have the lowest number of flights to offer.

- Regarding the direct flights, we have around 96,63% (4543) of the flights are direct flights and only 158 flights are fflights which have 1 stay in between flights.

- Barcelona airport is the airport where most airlines departs. And BVA Beauvais-Tille is the airport where less flights departs.

- Same happens with arrivals. Barcelona gets the most of the arrivals and BVA Beauvais-Tille gets the lowest numbers of arrivals.

- Regarding Price, we have range from 1 to 2704 euros.

- Departure day, we only have 1 day, which is 26th.

- Arrival day, we have 2 days, 5th and 2nd.

- Departure and Arrival month, we have 2 months, March and April.

- Departure and Arrival Year, only one, 2022.

- regarding Departure Time Hours, we have range from 7am which has around 809 flights offer to 8h which has only 6 offers. We also have nights flights like 21h which has 56 flights.

- regarding Departure Time Minutes, 00 minutes has the highest number of flights and 40minutes has the lowest number of flights.

- regarding Arrival Time Hours, we have range from 8am which has around 873 flights offer to 16h which has only 7 offers. We also have nights flights arrivals like 23h which has 17 flights.

- regarding Arrival Time Minutes, 45 minutes has the highest number of flights and 30 minutes has the lowest number of flights.

- mostly travel hours are between 1 to 2 hours and travel minutes are from 40 to 50 minutes.

We can also mention that:

- we have 8 cases of flights being more than 600 euros.

- For majority of the cases we have the same airpots -BCN to ORLY or ORLY to BCN.

- As expected, we have all kind of airline companies.

- All of these flights take flight in February, interesting!! very interesting!

In addition, we have the following highlights:

- Vueling and combo of Ryanair and TransaviaFrance has the highest number of flights offer.

- Most of the flights are direct flights.

- The city that depart the most is Barcelona.

- The city that has most of the arrivals is also Barcleona

- We have one unique Departure Date which is 26th. The same happens with Year, unique value of 2022.

- The month we have are February and March. February has more flights than March.

- As we said, when comparing ArrDate and ArrMonth, we see a tendency to have more flights near to current date than far away.

- When comparing Dep Time Hour and Mint, we see hours like 7am,9am,13h and 12h have more probability to have flights available than the remainign departures timing. And regarding the Dep Time Mint, we have that most of the flights departs at minutes 00, 50, 35, and 25. And the timing where less flights are offered tend to be at 16, 10, and 8am for Paris-Barcelona.

- And regarding the arrival timings, most of the flight offers tend to be most probable at 8am and less probable to be at 23, 18, or even 16h. And when comparing the minutes, we have most of the flights to be at 45min 25min, 35min and 05minutes.

- When comparing Travel_hours, we see it is most probably to be between 1h and 2h. There is way less flights which travel duration is more than 2h.

- When comparing Travel_minutes, we see it is most probably to be between 40min and 50min. There is way less flights which travel duration minutes ends in 15minutes or 25 minutes.

When talking about the modelization, we have that:

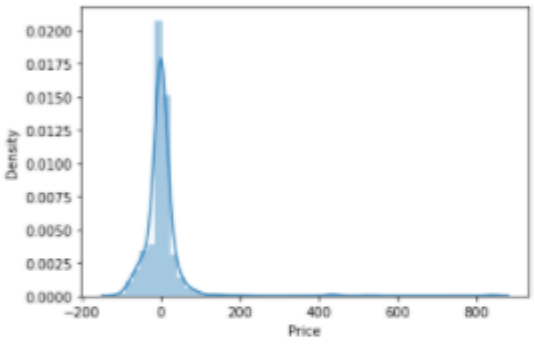
I considered all the results and scores one by one and compared this way all algorithms and chose the best one which is LASSO:

```
print('MSE:', metrics.mean_squared_error(y_test[0:1176], yhatls))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test[0:1176], yhatls))) #LOWEST MEAN ERRORS:
```

```
MAE: 23.26274597946242
MSE: 3173.9016145319465
RMSE: 56.33739090987394
```

[108]:
```
#prediction on the test data and check residual:
prediction=ls.predict(X_test)
sns.distplot(Y_test-prediction)
```

[108]:
```
<AxesSubplot:xlabel='Price', ylabel='Density'>
```



[112]:
```
#r2 score of the predicted values of training values compared to actual price:
result_ontestfromtrainingset=r2_score(Y_test,prediction)
abs(result_ontestfromtrainingset)                    #BESTEST Result in this case:
```

[112]:
```
0.06068094429588089
```

[113]:
```
#saving the model: #we will FINALLY SAVE this model as the final model because we have no more time to stu
import pickle
filename='Flight.pkl'
pickle.dump(ls,open(filename,'wb'))
```

[114]:
```
#Loading the model and checking the accuracy on the test data:
import pickle
loaded_model=pickle.load(open('Flight.pkl','rb'))
result=loaded_model.score(X_test,Y_test)
print(result)
```

```
0.030056939714088182
```

You can easily understand the all the process I did when you read my python script as it has complete similar comments on most of the main steps and everything written here will make more sense and you will understand all the steps and considerations commented here. Please go through all the comments in the given code script and will also make it easy to understand this report.

Let's check and revise the main findings and highlight the most highlightable results we got in between the whole process:

As we said, we tried different algorithms which were:

- LinearRegression
- DecisionTreeRegressor
- KNeighborsRegressor
- SVR
- Lasso
- RFR
- AdaBoostRegressor
- GradientBoostingRegressor

As we said previously, we tried different algorithms through different ways including using ElasticNet, regulating LR through Lasso, and even evaluating model performance with outliers removed using isolation forest, elliptical envelope, local outlier factor LOF, one class SVM, etc.

After checking and comparing the results, we can highlight the following ones as the best ones through comparison of r2score and the 3-error metrics:

```
[101]: x_train=X_train
       y_train=y_train
       x_test=X_test
       model.fit(x_train,y_train)
       pred=model.predict(x_test)
       test_score=r2_score(y_test,pred)
       train_score=r2_score(y_train,model.predict(x_train))
       if abs(train_score-test_score)<=0.3:
           print(model)
           print(abs(train_score-test_score))
           print("R2 score is: ", r2_score(y_test,pred))
           print("R2 score for train data: ", r2_score(y_train,model.predict(x_train)))
           print("Mean absolute error is: ", mean_absolute_error(y_test, pred))
           print("Mean squared error is: ", mean_squared_error(y_test, pred))
           print("Root mean squared error is: ", (mean_squared_error(y_test,pred)))

       LinearRegression()
       0.005578364089641075
       R2 score is:  0.02351419916815012
       R2 score for train data:  0.017935835078509044
       Mean absolute error is:  22.302395673754685
       Mean squared error is:  3000.4058698973145
       Root mean squared error is:  3000.4058698973145
```

[96]:
```
#Let's take radom state of 5

from sklearn.linear_model import LinearRegression
lr=LinearRegression()
X_train, X_test, Y_train, Y_test = train_test_split(x, Y, test_size=0.25, random_state=5)
lr.fit(X_train,Y_train)
```

[96]:
```
LinearRegression()
```

[102]:
```

from sklearn.linear_model import LinearRegression
lr=LinearRegression()
#X_train, X_test, Y_train, Y_test = train_test_split(x, Y, test_size=0.25, random_state=5)
lr.fit(X_train,y_train)
```

[102]:
```
LinearRegression()
```

[97]:
```
#checking the R2 Score for the linear regression:
from sklearn.metrics import r2_score
pred_test=lr.predict(X_test)
print(r2_score(Y_test,pred_test))
```

```
0.03127120643709047
```

[104]:
```
#checking the R2 Score for the linear regression:
from sklearn.metrics import r2_score
pred_test=lr.predict(X_test)
print(r2_score(y_test,pred_test))
```

```
0.02351419916815012
```

[ ]:
```
# evaluate the model thorugh mean absolute error
yhatlr = lr.predict(X_test)
# evaluate predictions
maelr = mean_absolute_error(y_test[0:1176], yhatlr)
print('MAE: %.3f' % maelr)
```

```
MAE: 23.508
```

[108]:
```

print(lr)
print("R2 score is: ", r2_score(y_test,yhatlr))
print("R2 score for train data: ", r2_score(y_train,lr.predict(x_train)))
print("Mean absolute error is: ", mean_absolute_error(y_test, yhatlr))
print("Mean squared error is: ", mean_squared_error(y_test, yhatlr))
print("Root mean squared error is: ", (mean_squared_error(y_test,yhatlr)))
```

```
LinearRegression()
R2 score is:  0.02351419916815012
R2 score for train data:  0.017935835078509044
Mean absolute error is:  22.302395673754685
Mean squared error is:  3000.4058698973145
Root mean squared error is:  3000.4058698973145
```

[85]:
```
#checking the r2 score with Lasso technique:
ls=Lasso(alpha=1,random_state=0)
ls.fit(X_train,Y_train)
ls.score(X_train,Y_train)
pred_ls=ls.predict(X_test)
lss=r2_score(Y_test,pred_ls)
lss
```

[85]:
```
0.024420237647289067
```

[220]:
```
### Splitting Dataset ###

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y , test_size = 0.2, random_state = 42)
```

[221]:
```
#Model Training
### Simple Linear Regression:

from sklearn.linear_model import LinearRegression
regressor = LinearRegression( fit_intercept = True)

regressor.fit(X_train, y_train)
```

[221]:
```
LinearRegression()
```

[222]:
```
print(f"Linear coefficients : {regressor.coef_}")
print(f"Intercept : {regressor.intercept_}")
```

```
Linear coefficients : [-1.24639421e-01  2.33915873e-01  1.57328521e+00  3.26736975e+00
  1.19939705e+00 -8.88178420e-15 -1.57801658e+01 -3.10862447e-15
  1.77635684e-15  8.82620712e-02 -2.02666337e-01 -2.15200017e-01
  2.61763564e+00]
Intercept : 133.77447047202958
```

[223]:
```
#Model Prediction
y_pred = regressor.predict(X_test)
```

[224]:
```
#Metrices: RMSE

from sklearn import metrics
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Root Mean Squared Error: 47.58332599935789
```

[225]:
```
# R-squared

from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

[225]:
```
0.025973293513137152
```

[304]:
```
#till now best cross fold is 8
cv_score=cross_val_score(ls,X_train,Y_train,cv=8)
cv_mean=cv_score.mean()
cv_mean                                #best cv mean till now
```

[304]:
```
0.026677000458969313
```

[105]:
```
cv_model = ElasticNetCV(cv = 10).fit(X_train,Y_train)

# without Lambdas, what's the alpha?

cv_model.alpha_

#Accordingly, we find the alpha value as 0.029719762751637686. Afterward, we can find the constant of the model establishe
```

[105]:
```
0.029719762751637686
```

[162]:
```
#Let's take radom state of 5

from sklearn.linear_model import LinearRegression
lr=LinearRegression()
X_train, X_test, Y_train, Y_test = train_test_split(x, Y, test_size=0.25, random_state=5)
lr.fit(X_train,Y_train)
```

[162]:
```
LinearRegression()
```

[163]:
```
#checking the R2 Score for the Linear regression:
from sklearn.metrics import r2_score
pred_test=lr.predict(X_test)
print(r2_score(Y_test,pred_test))
```

```
0.03127120643709047
```

```
parameters={'criterion':['mse','mae'],'max_features':["auto","sqrt","log2"]}
rf=RandomForestRegressor()
clf=GridSearchCV(rf,parameters)
clf.fit(X_train,Y_train[0:3149])
print(clf.best_params_)
```

```
{'criterion': 'mae', 'max_features': 'sqrt'}
```

[ ]:
```
rf=RandomForestRegressor(criterion="mae",max_features="sqrt")
rf.fit(X_train,Y_train[0:3149])
rf.score(X_train,Y_train[0:3149])
pred_decision=rf.predict(X_test)

rfs=r2_score(Y_test,pred_decision[0:1176])
print('R2 score:',rfs*100)

rfscore=cross_val_score(rf,x,Y, cv=5)
rfc=rfscore.mean()
print('Cross Val Score:', rfc*100)

pred_test=rf.predict(X_test)
print(r2_score(Y_test,pred_test))
```

```
R2 score: -0.12046261054916574
```

[ ]:
```
#In order to validate the model, Let's see if the mean errors are less so the model is predicting with high performance
from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

[167]:
```
import pickle
filename='Aba.pkl'
pickle.dump(rf,open(filename,'wb'))
```

[168]:
```
#Conslusion
import pickle
loaded_model=pickle.load(open('Aba.pkl','rb'))
result=loaded_model.score(X_test,Y_test)
print(result)
```

```
-0.03931548287893927
```

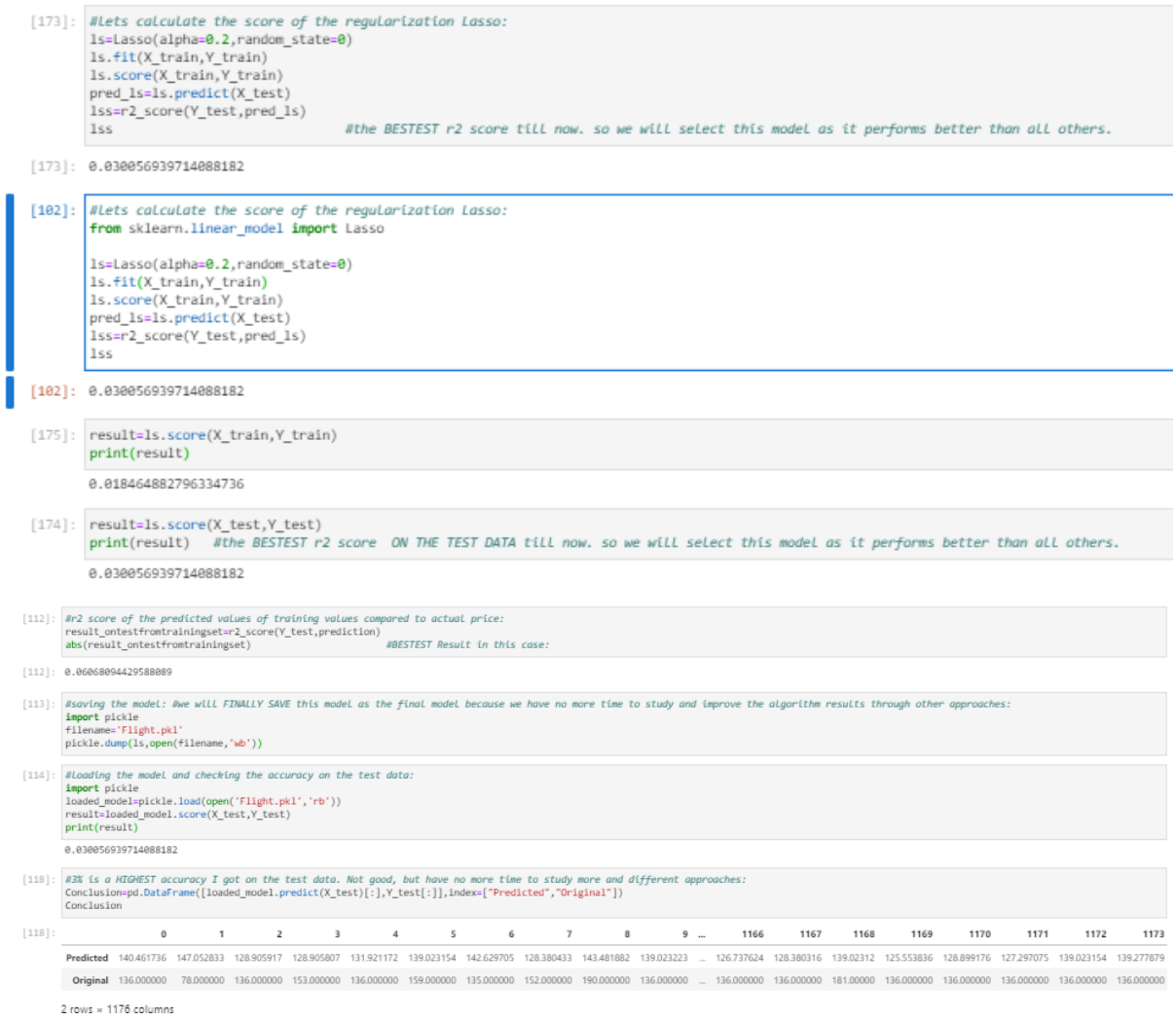
[283]:
```
# Now Let's use our model to make predictions on test data

# Creating X_test_new dataframe by dropping variables from X_test
X_test_new = X_test

# Adding a constant variable
X_test_new = sm.add_constant(X_test_new)

# Making predictions
y_pred = lm.predict(X_test_new)
```

[284]:
```
#check r2_score:
from sklearn.metrics import r2_score
r2_score(y_true = Y_test, y_pred = y_pred)
```

[284]:
```
0.033384255808418195
```

•[170]:
```
#training with Lasso
ls=Lasso(alpha=1,random_state=0)
ls.fit(X_train,Y_train)
ls.score(X_train,Y_train)
pred_ls=ls.predict(X_test)
lss=r2_score(Y_test,pred_ls)
lss  #So LASSO is our final selected model.
```

[170]:
```
0.024420237647289067
```



```
[173]: #Lets calculate the score of the regularization Lasso:
ls=Lasso(alpha=0.2,random_state=0)
ls.fit(X_train,Y_train)
ls.score(X_train,Y_train)
pred_ls=ls.predict(X_test)
lss=r2_score(Y_test,pred_ls)
lss                              #the BESTEST r2 score till now. so we will select this model as it performs better than all others.

[173]: 0.030056939714088182

[102]: #Lets calculate the score of the regularization Lasso:
from sklearn.linear_model import Lasso

ls=Lasso(alpha=0.2,random_state=0)
ls.fit(X_train,Y_train)
ls.score(X_train,Y_train)
pred_ls=ls.predict(X_test)
lss=r2_score(Y_test,pred_ls)
lss

[102]: 0.030056939714088182

[175]: result=ls.score(X_train,Y_train)
print(result)

0.018464882796334736

[174]: result=ls.score(X_test,Y_test)
print(result)   #the BESTEST r2 score  ON THE TEST DATA till now. so we will select this model as it performs better than all others.

0.030056939714088182

[112]: #r2 score of the predicted values of training values compared to actual price:
result_ontestfromtrainingset=r2_score(Y_test,prediction)
abs(result_ontestfromtrainingset)                    #BESTEST Result in this case:

[112]: 0.06068094429588089

[113]: #saving the model: #we will FINALLY SAVE this model as the final model because we have no more time to study and improve the algorithm results through other approaches:
import pickle
filename='Flight.pkl'
pickle.dump(ls,open(filename,'wb'))

[114]: #Loading the model and checking the accuracy on the test data:
import pickle
loaded_model=pickle.load(open('Flight.pkl','rb'))
result=loaded_model.score(X_test,Y_test)
print(result)

0.030056939714088182

[118]: #3% is a HIGHEST accuracy I got on the test data. Not good, but have no more time to study more and different approaches:
Conclusion=pd.DataFrame([loaded_model.predict(X_test)[:],Y_test[:]],index=["Predicted","Original"])
Conclusion
```

[118]:

	0	1	2	3	4	5	6	7	8	9	...	1166	1167	1168	1169	1170	1171	1172	1173
Predicted	140.461736	147.052833	128.905917	128.905807	131.921172	139.023154	142.629705	128.380433	143.481882	139.023223	...	126.737624	128.380316	139.02312	125.553836	128.899176	127.297075	139.023154	139.277879
Original	136.000000	78.000000	136.000000	153.000000	136.000000	159.000000	135.000000	152.000000	190.000000	136.000000	...	136.000000	136.000000	181.00000	136.000000	136.000000	136.000000	136.000000	136.000000

2 rows × 1176 columns

This last is the BESTEST Score we got on the training and also on the testing set. So, even after this final selected model, we tried to handle the outliers and did caping but the results did not improve. So, this last screenshot's saved model is the final one.

CONCLUSION

- Key Findings and Conclusions of the Study

If we concentrate on the main analysis and conclusions, we can say:

- Question 1 –Which variables are important to predict the price of flight in our dataset?

If we first concentrate on the target and check the best correlations against it, then we will see that the following features are considered the best correlated features within our dataset:

Regarding the target Price, we see the features that are highly correlated with it are the Departure and Arrival Date.

We also see that:

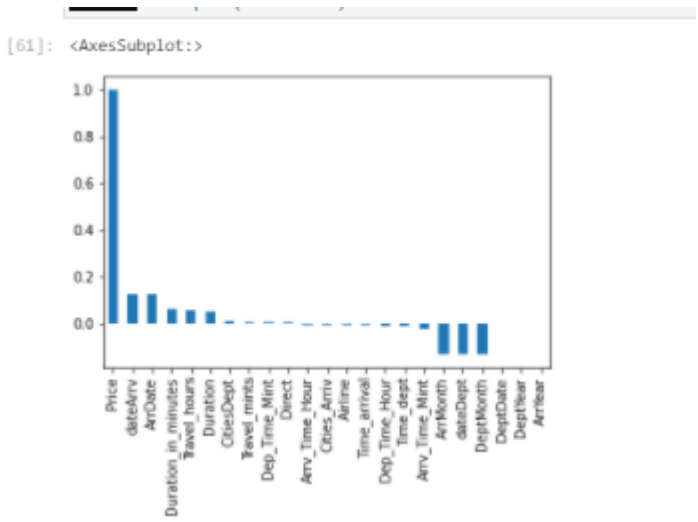
- Departure city and Arrival city are highly correlated with Duration and Arriva Time.

- We also see the departure City and arrival city are also very highly correlated.

- We see departure time is almost fully correlated with departure time hour. And same happens with Arrival case, we see arrival time is almost fully correlated with Arrival time hour.

- And as it is expected, travel hours is almost fully correlated with duration in minutes and the same time these are almost fully correlated with duration.

And if we check the highest correlated features with regards the target Price, we see that



Price 1.000000

dateArrv 0.128731

ArrDate 0.128731

ArrMonth -0.128731

dateDept -0.128731

DeptMonth -0.128731

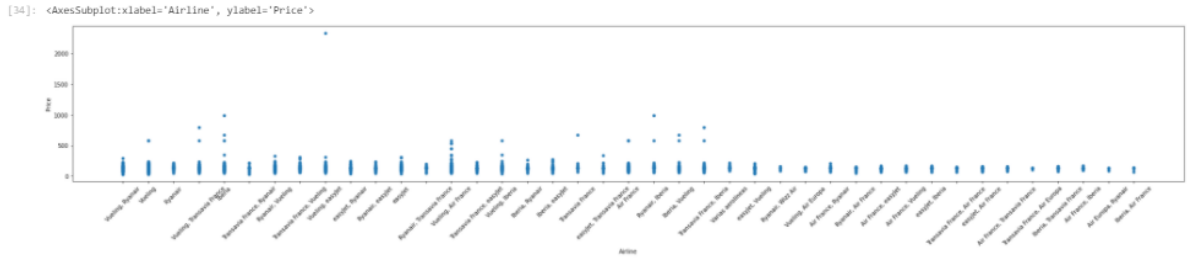
Name: Price, dtype: float64

As we said, we have departure date, Arrival date as the main 2 independent features that impacts the most on the Price of the flights.

And if we try to answer the following question after having worked on this project, we see that:

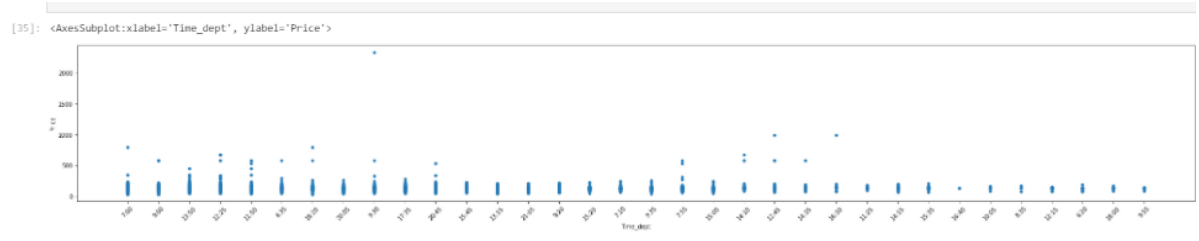
- Do airfares change frequently? Yes, they do.
- Do they move in small increments or in large jumps? Regarding Paris-Bcn flights, as we mentioned throughout the project, the average price of airfare is around and between 0 to 250 euros per head. So, since the price starts from 29 euros and then, we see the price goes up. When we changed the journey date from March to April, the ticket average seems not to have changed too much. So, we can say the changes on small prices are small.
- Do they tend to go up or down over time? As we said, the prices for Barcelona and Paris flights haven't changed too much overtime as average. There is a competition for these kind of flights like Paris-Bcn flights as these airports have large number of airlines and the traffic and competition between airlines is huge.
- What is the best time to buy so that the consumer can save the most by taking the least risk? There is no best time as the prices changes overtime and it also depends on other factors ticket change or special offers. Also, in our case, there was only 2 different periods for which the average airfare hasn't change too much.
- Does price increase as we get near to departure date? Not always as in our case it did not happen at average. For example, we had the same lowest price offer for Feb-March flight and also the same price lowest price for March-April month.

Is Vueling cheaper than Ryanair?

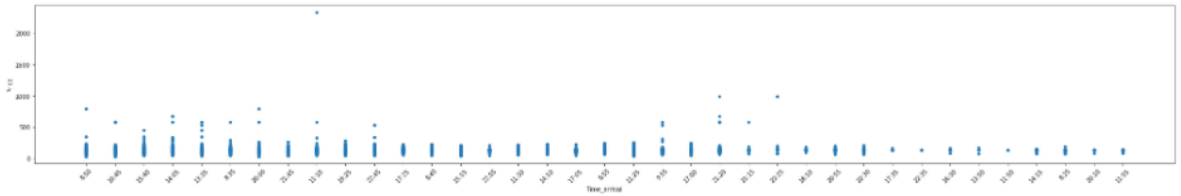


We see almost all airlines have similar median flights prices except few ones which sometimes have higher prices. These exceptions occurr with airlines like flights that are combined such like Vueling, Transavia France Iberia and also with Vueling and Ryanair or even when taking flights with Ryanair and Transvia France. We also have few higher prices when combining airlines like Easyjet, Transavia France and Air France. Also few higher flights price when taking flights with Ryanair and Iberia or even Iberia and Vueling.

Are morning flights expensive?



Not all the time, regarding the departure time, we have few timings that may have higher prices than the mean and which are flights departing at 7am, 9am, 13:50pm, 12:25 pm, 11:50am, 18:20, 9:30am, 20:45, 7:55am, 14:10, 12:45pm, 14:35 and 16:30h.



Similar situation happens with Arrival Time, not always morning flights are expensive. We have Price around the mean and the median but we have also Arrival Time where we have a bit higher prices such as in timings like 8:50, 10:45 (one case), 15:40, 14:05, 13:35,8:35 (one case), 20h (2 cases), 21:45, 11:10, 22:45 (2 cases), 9:55, 21:20, 21:15 (one ase) and 23:25 (one case).

- # Learning Outcomes of the Study in respect of Data Science

In this study, our models are trained with flight price data using Linear Regression, Random Forest Regressor, SVR, GBR, Adaboost among others. We have saw that advanced machine learning algorithms got very low accurate r2 scores and testing scores for prediction of airfares prices, as evaluated by the performance metrics. Given our dataset used for this study, our main conclusion is that Linear Regression can give us way better results through Lasso regularization and after having handled outliers through for example capping or technique likes Isolation forest, elliptical envelope, local outlier factor or even through One class SVM which are able to generate comparably accurate price estimations with lower prediction errors, compared with the normal base ML like Linear Regression or even Random Forest Regressor results.

The existing literature has demonstrated mixed results about which algorithm is superior in making predictions. Previous studies suggest that RF has a better performance than GBR but other studies demonstrate that both models have comparable predictive power and almost equally applicable in making predictions (Ahmad et al., 2017). In another study about the forecast of daily lake water levels, RF is found to exhibit the best results when compared to linear regression, SVM and ANN with respect to R2R2 and RMSE (Li et al., 2016). Utilising RF, stochastic gradient boosting and SVM to predict genomic breeding values, Ogutu et al. (2011) conclude that boosting has had the best performance, followed by SVM and RF. In this study, SVM performs better than RF (see also Caruana & Niculescu-Mizil, 2006).

I think we should suggest that the choice of algorithm depends on several factors including, the size of data set, computing power, time constraint and researcher's knowledge about machine learning and obvious, the useful FEATURES with regards the target VARIBLE PRICE. If the accuracy of prediction is the priority, we recommend utilizing LinearRegression with Lasso Regularization and also due to that we could see that Lasso improved the results of Linear Regression in our case. Needless to say, it is a good practice to use more than one algorithm to compare the predictions.

- Limitations of this work and Scope for Future Work

What are the limitations of this solution provided, the future scope? What all steps/techniques can be followed to further extend this study and improve the results.

Our analysis result showed that this research area has not been greatly explored throughout the project and that there still exist several aspects which need to be properly and thoroughly investigated including performance issues, dataset issues, usage of dynamic external features such as social media data and most importantly THE IMPORTANT FEATURES FOR OUR TARGET.

In addition, we suggest a deep learning and social media data based prediction model as the one of the most promising avenues of research going forward. We say to add social media as the airline ticket price could be influenced by several dynamic factors which can be captured from social media data. This include occurrence of some event at the origin or destination city as mentioned earlier.

However, as far as our knowledge is concerned, there is no existing work that utilizes social media data to predict demand and or ticket prices. Motivated by previous studies, we can think of various additional useful features from social media that can possibly forecast airlines passenger demand and or ticket prices. For example, sentiment analysis of different twitter hash tags could convey the presence of some event at a flight origin/destination city that improves the prediction of ticket price/demand.

Even, if we try the algorithms used in this project, we need to mention, that having important features and right techniques and managing the outliers, we can reach to a good performance of the models and get better accuracies and error metrics results.

Ok.