Project02_Revised_Final

March 24, 2022

1 Project 2

1.0.1 CS 5/7394 - Applied Machine Learning

- **Due** - March 11 @ 11:59 pm pushed to Github repo
- **Teams** - You can do this project solo or in pairs. Not 3, not 4 not 5… Max of 2. If a 5394 student pairs with a 7394 student, the pair needs to do the 7394 work.

Below are 6 Kaggle Datasets. You will choose 1 to work with for this project.

- Airfare Prediction Dataset
- Chinese Rest Holiday Dataset
- Jigsaw Toxic Comment Classification Challenge
- Latest Covid 19 Dataset Worldwide
- Trains
- Football Data top 5 Leagues

Merging disparate datasets is a staple of the data exploration process. Therefore, for which ever data set above that you choose, you will need to independently find **an additional** dataset to merge with your selection. The only requirement is that it add to the richness of the original dataset. Students in the 7000-level version of the class need to find two additional data sets to merge with the original selection.

Note: If you want to start with a different data set, you need to get Fontenot's OK first.

1.0.2 Your Tasks

Below, there are cells that provide directions on what to do for the project.

You can insert as many cells between the ones below as you'd like, but please **Do NOT** change the cells already provided.

1.0.3 Part 1 - Getting Started

- Import libraries
- Load original Data (which ever one you chose from the provided list) into a data frame.
- Load your additional data set(s) into a data frame.
- In a markdown cell, provide a brief description of your the data sets you've chosen to work with.

- Develop a list of 3 - 4 questions that you hope to be able to answer after the exploration of the data and write them in this section.

1.0.4 Datasets being used:

- Airfare Prediction Dataset
- Busiest Airports
- Jet Fuel Prices

[1]:
```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
from IPython.display import Markdown as md
import warnings

warnings.filterwarnings('ignore')

airfare_prediction = pd.read_excel('data/train.xlsx')
jet_fuel = pd.read_csv('data/jet_fuel_data.csv')
busiest_airports = pd.read_csv('data/BusiestAirports_2019.csv',␣
 ↪encoding='utf-8')
```

[2]:
```
airfare_prediction.head()
```

[2]:

	Airline	Date_of_Journey	Source	Destination	Route	\
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	

	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional Info	Price
0	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3898
1	05:50	13:15	7h 25m	2 stops	No info	7663
2	09:25	04:25 10 Jun	19h	2 stops	No info	13883
3	18:05	23:30	5h 25m	1 stop	No info	6219
4	16:50	21:35	4h 45m	1 stop	No info	13303

Airfare Prediction Dataset Description: Dataset that contains information on various flights from various airlines to help predict price of ticket.

[3]:
```
jet_fuel.head()
```

[3]:

	Date	Price	Change
0	Feb-17	1.55	-
1	Mar-17	1.45	-6.59%
2	Apr-17	1.51	4.50%
3	May-17	1.41	-6.49%
4	Jun-17	1.30	-8.29%

Busiest Airports Dataset Description: Dataset that uses the total number of passengers that

pass through an airport to determine the ranking for busiest airports.

[4]:
```
busiest_airports.head()
```

[4]:
```
   Rank                                    Airport  \
0     1  Hartsfield-Jackson Atlanta International Airport
1     2             Beijing Capital International Airport
2     3                 Los Angeles International Airport
3     4                       Dubai International Airport
4     5                              Tokyo Haneda Airport

                  Location              Country Code(IATA/ICAO)  \
0          Atlanta, Georgia        United States        ATL/KATL
1  Chaoyang-Shunyi, Beijing                China        PEK/ZBAA
2   Los Angeles, California        United States        LAX/KLAX
3            Garhoud, Dubai  United Arab Emirates        DXB/OMDB
4                Ōta, Tokyo                Japan        HND/RJTT

  Total Passengers
0      110,531,300
1      100,011,438
2       88,068,013
3       86,396,757
4       85,505,054
```

Jet Fuel Dataset Description: Dataset that outlines price of jet fuel per gallon each month starting in February 2019 and percent change from the price of the month before.

Questions we hope to answer:

1. What affect does the departure/arrival location have on airfare
2. What affect does the flight date/time have on airfare
3. What affect does the price of jet fuel have on airfare
4. What affect does the choice of airline have on airfare

1.0.5 Part 2 - Data Inspection

Write some code to summarize the datasets. Think about the following questions: - What type of data is each variable? (think like a data scientist here, not a computer scientist) - What is the total size of the data sets? - What time boundaries are there in the dataset? IOW, what time frame do they span? - Are there any missing values in any of the variables?

Do this with Intentionality. Don't skimp.

1.0.6 Airfare Prediction Dataset

Data Type for Each Variable: - Airline: object/string - Date_of_Journey: object/string converted to datetime object - Source: object/string - Destination: object/string - Route: object/string

- Dep_Time (Departing Time): object/string converted to datetime object - Arrival_Time: object/string converted to datetime object - Duration: object/string - Total_Stops: object/string - Additional Info: object/string - Price: float ***Total Size of Data Set:*** - 2671 entries

Time Boundaries:

[5]:
```
sorted_dates = pd.to_datetime(airfare_prediction["Date_of_Journey"], format =␣
↪"%d/%m/%Y").sort_values().dt.date
min = sorted_dates.min()
max = sorted_dates.max()
print(str(min) + " to " + str(max))
```

```
2019-03-01 to 2019-06-27
```

Missing Values:

[6]:
```
missing = airfare_prediction[airfare_prediction.isna().isnull().any(axis=1)]
missing
```

[6]:
```
Empty DataFrame
Columns: [Airline, Date_of_Journey, Source, Destination, Route, Dep_Time,
Arrival_Time, Duration, Total_Stops, Additional Info, Price]
Index: []
```

[7]:
```
airfare_prediction.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional Info  10683 non-null  object
 10  Price            10683 non-null  int64
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
```

1.0.7 Busiest Airports in 2019

Data Type for Each Variabe: - Rank: integer - Airport: object/string - Location: object/string - Country: object/string - Code(IATA/ICAO): object/string - Totalpassengers: object/string ***Total Size of Data Set:*** - 50 entries ***Time Boundaries:*** - N/A ***Missing Values:***

[8]:
```
missing = busiest_airports[busiest_airports.isna().isnull().any(axis=1)]
missing
```

[8]:
```
Empty DataFrame
Columns: [Rank, Airport, Location, Country, Code(IATA/ICAO), Total Passengers]
Index: []
```

[9]:
```
busiest_airports.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Rank              50 non-null     int64
 1   Airport           50 non-null     object
 2   Location          50 non-null     object
 3   Country           50 non-null     object
 4   Code(IATA/ICAO)   50 non-null     object
 5   Total Passengers  50 non-null     object
dtypes: int64(1), object(5)
memory usage: 2.5+ KB
```

1.0.8 Jet Fuel Dataset

Data Type for Each Variable: - Month: object/string - Price: object/string - Change: object/string ***Total Size of Data Set:*** - 60 entries

Time Boundaries:

[10]:
```
sorted_dates = pd.to_datetime(jet_fuel["Date"], format = "%b-%y").sort_values().
↪dt.date

min = str(sorted_dates.min().month) + "-" + str(sorted_dates.min().year)
max = str(sorted_dates.max().month) + "-" + str(sorted_dates.max().year)
print(min + " to " + max)
```

```
2-2017 to 1-2022
```

Missing Values:

[11]:
```
missing = jet_fuel[jet_fuel.isna().isnull().any(axis=1)]
missing
```

[11]:
```
Empty DataFrame
Columns: [Date, Price, Change]
Index: []
```

[12]:
```
jet_fuel.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60 entries, 0 to 59
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    60 non-null     object
 1   Price   60 non-null     float64
 2   Change  60 non-null     object
dtypes: float64(1), object(2)
memory usage: 1.5+ KB
```

1.0.9 Part 3 - Data Description

- Create a data description (data dictionary) for your data sets.
  - Describe each variable
  - If categorical, what levels are present? If the levels are encoded, what do the codes mean?
  - If numeric, provide min, max, median and any other univariate stats you'd like to add in.
- Where appropriate, provide histograms or other visualizations to characterize each variable.

1.0.10 Airfare Prediction Dataset

Variable Name	C/N	Description
Airline	Categorical	The name of the airline carrier, if multiple value = "Multiple carriers"
Date_of_Journey	Categorical	Date of the departure time (dd/mm/yyyy)
Source	Categorical	The flight's city of departure
Destination	Categorical	The flight's city of arrival
Route	Categorical	List of airport stops in order
Dep_Time	Categorical	Time of departure (24 hour format)
Arrival_Time	Categorical	Time of arrival (24 hour format)
Duration	Categorical	Length of flight in hours and minutes
Total_Stops	Categorical	Number of stops made in the flight, if 0 value is "non-stop"
Additional_Info	Categorical	Other information needed to be listed
Price	Numerical	Price of ticket

1.0.11 Busiest Airports

Variable Name	C/N	Description
Rank	Categorical	Order of airport by level of busyness
Airport	Categorical	Full name of the airport

Variable Name	C/N	Description
Code (IATA/ICAO)	Categorical	Airport codes for the International Air Transport Association (IATA) and the Internation Civil Aviation Organization (ICAO)
Location	Categorical	Where the airport is located
Country	Categorical	Country the airport is in
Dep_Time	Categorical	Time of departure (24 hour format)

1.0.12 Jet Fuel

Variable Name	C/N	Description
Month	Categorical	Month and Year of when the data was taken
Price	Numerical	How much jet fuel costs per gallon
Change	Numerical	Percent change of jet fuel price from the month before

[38]:
```
# Getting min, max, median of price and change
jet_fuel_norm = jet_fuel[jet_fuel.Change != "-"]
jet_fuel_chg = (jet_fuel_norm['Change'].str.rstrip('%').astype('float') / 100.
 ↪0).sort_values()
jet_fuel_dates = pd.to_datetime(jet_fuel_norm["Date"], format = "%b-%y").
 ↪sort_values()

c_min = jet_fuel_chg.min()
c_max = jet_fuel_chg.max()
c_median = jet_fuel_chg.median()

p_min = jet_fuel_norm["Price"].min()
p_max = jet_fuel_norm["Price"].max()
p_median = jet_fuel_norm["Price"].median()

md("| Variable Name | Min | Max | Median |\n" +
   "| :-----------: | :-: | :-: | :----: |\n" +
   "| Change | {} | {} | {} |\n".format(c_min, c_max, c_median) +
   "| Price | {} | {} | {} |".format(p_min, p_max, p_median))
```

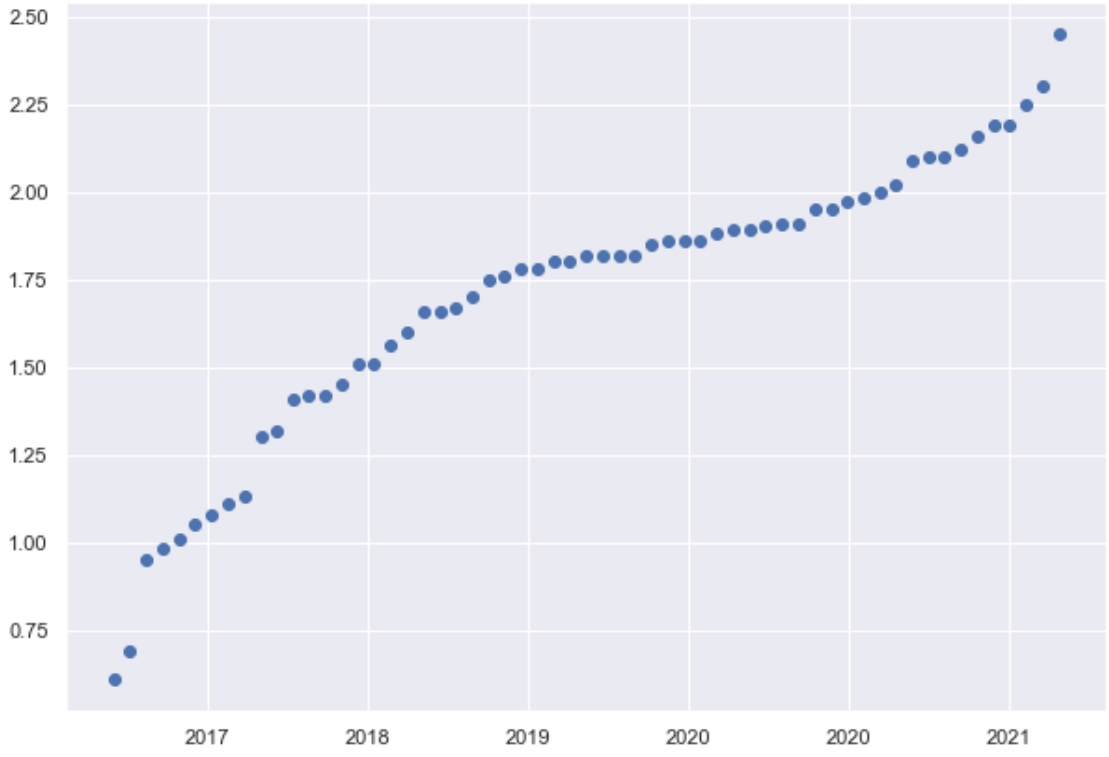
[38]:

Variable Name	Min	Max	Median
Change	-0.3685	0.4329	0.032400000000000005
Price	0.61	2.45	1.82

Price per Month

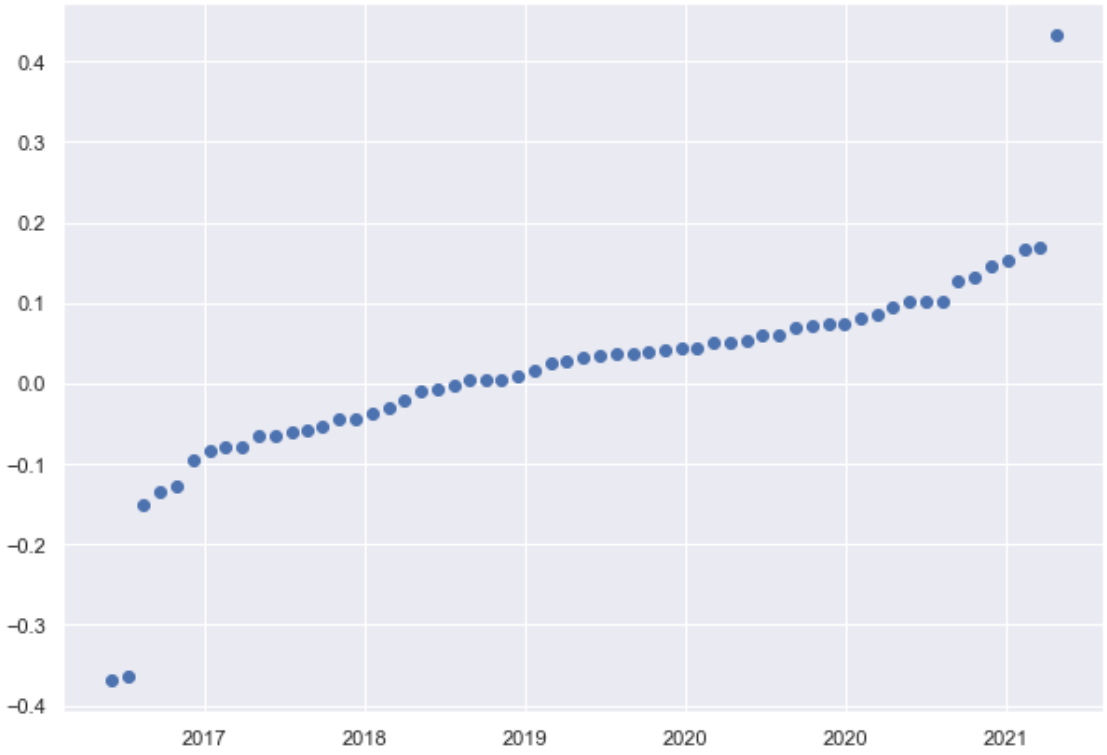
[41]:
```
axes = plt.axes()
axes.xaxis.set_major_locator(plt.MaxNLocator(7))
```

```
plt.scatter(jet_fuel_dates, jet_fuel_norm["Price"].sort_values())
plt.show()
```



Price Change per Month

[42]:
```
axes = plt.axes()
axes.xaxis.set_major_locator(plt.MaxNLocator(7))
axes.yaxis.set_major_locator(plt.MaxNLocator(10))
plt.scatter(jet_fuel_dates, jet_fuel_chg)
plt.show()
```



1.0.13 Part 4 - Merge the data

Now that you have a better feel for each of your two (or three, for the 7394 students) data sets, it is time to merge them. Describe your strategy for merging the data sets and then actually perform the merge.

Develop a strategy for verifying that the data is properly merged (hoping and finger-crossing are not valid strategies).

description of merge and validation strategy We will merge the additional two datasets – jet fuel and busiest airports – into the main dataset in similar but independent ways. For the busiest airports dataset, we will focus on the source and destination airports and merge these two datasets using two columns – one storing the rank of the source airport and the other storing the rank of the destination airport. To merge the jet fuel dataset into the original airfare prediction dataset, we will create a new column that stores the associated jet fuel price associated with the month and year of the flight. To validate that the datasets were successfully merged, we will output the dataframe and observe a few sample datapoints

[43]:
```
merged_data = airfare_prediction.copy(deep = True)
```

[44]:
```
merged_data
```

[44]:
```
            Airline Date_of_Journey    Source Destination  \
0            IndiGo      24/03/2019  Banglore   New Delhi
1          Air India       1/05/2019   Kolkata    Banglore
2        Jet Airways       9/06/2019     Delhi      Cochin
3            IndiGo      12/05/2019   Kolkata    Banglore
4            IndiGo      01/03/2019  Banglore   New Delhi
…               …             …         …          …
10678      Air Asia       9/04/2019   Kolkata    Banglore
10679    Air India       27/04/2019   Kolkata    Banglore
10680  Jet Airways       27/04/2019  Banglore       Delhi
10681      Vistara       01/03/2019  Banglore   New Delhi
10682    Air India       9/05/2019     Delhi      Cochin

                        Route Dep_Time  Arrival_Time Duration Total_Stops  \
0                   BLR → DEL    22:20  01:10 22 Mar   2h 50m    non-stop
1      CCU → IXR → BBI → BLR     05:50         13:15   7h 25m     2 stops
2      DEL → LKO → BOM → COK     09:25  04:25 10 Jun      19h     2 stops
3            CCU → NAG → BLR     18:05         23:30   5h 25m      1 stop
4            BLR → NAG → DEL     16:50         21:35   4h 45m      1 stop
…                        …       …            …       …          …
10678            CCU → BLR     19:55         22:25   2h 30m    non-stop
10679            CCU → BLR     20:45         23:20   2h 35m    non-stop
10680            BLR → DEL     08:20         11:20       3h    non-stop
10681            BLR → DEL     11:30         14:10   2h 40m    non-stop
10682  DEL → GOI → BOM → COK     10:55         19:15   8h 20m     2 stops

      Additional Info  Price
0             No info   3898
1             No info   7663
2             No info  13883
3             No info   6219
4             No info  13303
…               …     …
10678         No info   4108
10679         No info   4146
10680         No info   7230
10681         No info  12649
10682         No info  11754

[10683 rows x 11 columns]
```

[18]:
```
merged_data['Source Airport Busyness Rank'] = ""

for sourceIndex, source in enumerate(merged_data['Source']):
    airportFound = False
    for locIndex, location in enumerate(busiest_airports['Location']):
        city = location.split(',')
```

```
        if source == city[0]:
            airportFound = True
            merged_data['Source Airport Busyness Rank'].iloc[sourceIndex] =␣
↪busiest_airports['Rank'].iloc[locIndex]

    # if airport not found in dataset, set as null
    if airportFound == False:
        merged_data['Source Airport Busyness Rank'].iloc[sourceIndex] = np.nan
```

[19]:
```
merged_data
```

[19]:
```
           Airline Date_of_Journey    Source Destination  \
0           IndiGo      24/03/2019  Banglore   New Delhi
1        Air India       1/05/2019   Kolkata    Banglore
2      Jet Airways       9/06/2019     Delhi      Cochin
3           IndiGo      12/05/2019   Kolkata    Banglore
4           IndiGo      01/03/2019  Banglore   New Delhi
…              …             …         …          …
10678     Air Asia       9/04/2019   Kolkata    Banglore
10679    Air India      27/04/2019   Kolkata    Banglore
10680  Jet Airways      27/04/2019  Banglore       Delhi
10681      Vistara      01/03/2019  Banglore   New Delhi
10682    Air India       9/05/2019     Delhi      Cochin

                      Route Dep_Time  Arrival_Time Duration Total_Stops  \
0                   BLR → DEL    22:20  01:10 22 Mar   2h 50m    non-stop
1      CCU → IXR → BBI → BLR    05:50         13:15   7h 25m     2 stops
2      DEL → LKO → BOM → COK    09:25  04:25 10 Jun      19h     2 stops
3            CCU → NAG → BLR    18:05         23:30   5h 25m      1 stop
4            BLR → NAG → DEL    16:50         21:35   4h 45m      1 stop
…                        …       …             …       …         …
10678            CCU → BLR    19:55         22:25   2h 30m    non-stop
10679            CCU → BLR    20:45         23:20   2h 35m    non-stop
10680            BLR → DEL    08:20         11:20       3h    non-stop
10681            BLR → DEL    11:30         14:10   2h 40m    non-stop
10682  DEL → GOI → BOM → COK    10:55         19:15   8h 20m     2 stops

      Additional Info  Price Source Airport Busyness Rank
0             No info   3898                          NaN
1             No info   7663                          NaN
2             No info  13883                           17
3             No info   6219                          NaN
4             No info  13303                          NaN
…                ...    …                            …
10678         No info   4108                          NaN
```

```
10679         No info   4146                              NaN
10680         No info   7230                              NaN
10681         No info  12649                              NaN
10682         No info  11754                               17

[10683 rows x 12 columns]
```

[20]:
```
merged_data['Destination Airport Busyness Rank'] = ""

for sourceIndex, source in enumerate(merged_data['Destination']):
    airportFound = False
    for locIndex, location in enumerate(busiest_airports['Location']):
        city = location.split(',')

        if source == city[0]:
            airportFound = True
            merged_data['Destination Airport Busyness Rank'].iloc[sourceIndex]␣
 ↪= busiest_airports['Rank'].iloc[locIndex]

    # if airport not found in dataset, set as null
    if airportFound == False:
        merged_data['Destination Airport Busyness Rank'].iloc[sourceIndex] = np.
 ↪nan
```

[21]:
```
merged_data
```

[21]:
```
            Airline Date_of_Journey    Source Destination  \
0            IndiGo      24/03/2019  Banglore   New Delhi
1          Air India       1/05/2019   Kolkata    Banglore
2        Jet Airways       9/06/2019     Delhi      Cochin
3            IndiGo      12/05/2019   Kolkata    Banglore
4            IndiGo      01/03/2019  Banglore   New Delhi
…               …             …         …          …
10678      Air Asia       9/04/2019   Kolkata    Banglore
10679     Air India      27/04/2019   Kolkata    Banglore
10680   Jet Airways      27/04/2019  Banglore       Delhi
10681       Vistara      01/03/2019  Banglore   New Delhi
10682     Air India       9/05/2019     Delhi      Cochin

                      Route Dep_Time  Arrival_Time Duration Total_Stops  \
0                   BLR → DEL    22:20  01:10 22 Mar   2h 50m    non-stop
1       CCU → IXR → BBI → BLR    05:50         13:15   7h 25m     2 stops
2       DEL → LKO → BOM → COK    09:25  04:25 10 Jun      19h     2 stops
3             CCU → NAG → BLR    18:05         23:30   5h 25m      1 stop
4             BLR → NAG → DEL    16:50         21:35   4h 45m      1 stop
…                         …        …             …       …         …
10678               CCU → BLR    19:55         22:25   2h 30m    non-stop
```

```
10679              CCU → BLR    20:45         23:20   2h 35m    non-stop
10680              BLR → DEL    08:20         11:20      3h     non-stop
10681              BLR → DEL    11:30         14:10   2h 40m    non-stop
10682  DEL → GOI → BOM → COK    10:55         19:15   8h 20m     2 stops

      Additional Info  Price Source Airport Busyness Rank  \
0             No info   3898                           NaN
1             No info   7663                           NaN
2             No info  13883                            17
3             No info   6219                           NaN
4             No info  13303                           NaN
…                …    …                              …
10678         No info   4108                           NaN
10679         No info   4146                           NaN
10680         No info   7230                           NaN
10681         No info  12649                           NaN
10682         No info  11754                            17

      Destination Airport Busyness Rank
0                                   NaN
1                                   NaN
2                                   NaN
3                                   NaN
4                                   NaN
…                                   …
10678                               NaN
10679                               NaN
10680                                17
10681                               NaN
10682                               NaN

[10683 rows x 13 columns]
```

[22]:
```
merged_data['Jet Fuel Price at Date of Travel'] = ""

monthSwitcher = {
    "Jan": "01",
    "Feb": "02",
    "Mar": "03",
    "Apr": "04",
    "May": "05",
    "Jun": "06",
    "Jul": "07",
    "Aug": "08",
    "Sep": "09",
    "Oct": "10",
    "Nov": "11",
```

```
    "Dec": "12",
    }

yearSwitcher = {
    "17": "2017",
    "18": "2018",
    "19": "2019",
    "20": "2020",
    "21": "2021",
    "22": "2022"
}

for journeyDateIndex, journeyDate in enumerate(merged_data['Date_of_Journey']):
    journey_date = journeyDate.split('/')
    journey_month = journey_date[1]
    journey_year = journey_date[2]

    fuelPriceFound = False

    for fuelDateIndex, fuelDate in enumerate(jet_fuel['Date']):
        fuel_date = fuelDate.split('-')
        fuel_month = monthSwitcher[fuel_date[0]]
        fuel_year = yearSwitcher[fuel_date[1]]

#         print("Journey Month: ", journey_month, " Fuel Month: ", fuel_month)
#         print("Journey Year: ", journey_year, "Fuel Year: ", fuel_year)

        if journey_month == fuel_month and journey_year == fuel_year:
            fuelPriceFound = True
            merged_data['Jet Fuel Price at Date of Travel'].
 ↪iloc[journeyDateIndex] = jet_fuel['Price'].iloc[fuelDateIndex]

    if fuelPriceFound == False:
        merged_data['Jet Fuel Price at Date of Travel'].iloc[journeyDateIndex]␣
 ↪= np.nan
```

[23]:
```
merged_data
```

[23]:

	Airline	Date_of_Journey	Source	Destination	\
0	IndiGo	24/03/2019	Banglore	New Delhi	
1	Air India	1/05/2019	Kolkata	Banglore	
2	Jet Airways	9/06/2019	Delhi	Cochin	
3	IndiGo	12/05/2019	Kolkata	Banglore	
4	IndiGo	01/03/2019	Banglore	New Delhi	
…	…	…	…	…	
10678	Air Asia	9/04/2019	Kolkata	Banglore	
10679	Air India	27/04/2019	Kolkata	Banglore	

```
10680  Jet Airways      27/04/2019  Banglore       Delhi
10681      Vistara      01/03/2019  Banglore   New Delhi
10682    Air India       9/05/2019     Delhi      Cochin

                      Route Dep_Time  Arrival_Time Duration Total_Stops  \
0                 BLR → DEL    22:20  01:10 22 Mar   2h 50m    non-stop
1      CCU → IXR → BBI → BLR    05:50         13:15   7h 25m     2 stops
2      DEL → LKO → BOM → COK    09:25  04:25 10 Jun      19h     2 stops
3            CCU → NAG → BLR    18:05         23:30   5h 25m      1 stop
4            BLR → NAG → DEL    16:50         21:35   4h 45m      1 stop
…                      …       …             …       …          …
10678             CCU → BLR    19:55         22:25   2h 30m    non-stop
10679             CCU → BLR    20:45         23:20   2h 35m    non-stop
10680             BLR → DEL    08:20         11:20       3h    non-stop
10681             BLR → DEL    11:30         14:10   2h 40m    non-stop
10682  DEL → GOI → BOM → COK    10:55         19:15   8h 20m     2 stops

      Additional Info  Price Source Airport Busyness Rank  \
0             No info   3898                          NaN
1             No info   7663                          NaN
2             No info  13883                           17
3             No info   6219                          NaN
4             No info  13303                          NaN
…               …    …                              …
10678         No info   4108                          NaN
10679         No info   4146                          NaN
10680         No info   7230                          NaN
10681         No info  12649                          NaN
10682         No info  11754                           17

      Destination Airport Busyness Rank Jet Fuel Price at Date of Travel
0                                   NaN                             1.9
1                                   NaN                            1.97
2                                   NaN                            1.82
3                                   NaN                            1.97
4                                   NaN                             1.9
…                                     …                               …
10678                               NaN                            1.98
10679                               NaN                            1.98
10680                                17                            1.98
10681                               NaN                             1.9
10682                               NaN                            1.97

[10683 rows x 14 columns]
```

1.0.14 Part 5 - Explore Bivariate relationships

- Choose a reasoned set of variables to explore further. You don't have to explore all possible pairs of variables, nor do we want to grade that much. Choose 7 - 9 variables. One should be a variable that you'd like to predict (target variable) using the others (predictor variables).
- List your predictor variables
- List your target variable
- Briefly describe why you have chosen these.

Use any of the available visualizations from Seaborn to explore the relationships between the variables. Explore the relationships among the predictor variables as well as the relationship between each predictor variable and the target variable. Which of the predictor variables are most strongly related? Are there any interesting relationships between categorical predictors and numeric predictors? If there are any dichotomous variables, does that influence any of the relationships? Are the relationships positive or negative?

Below each plot, you should provide a description and interpretation of the plot. Make sure to include why the variables in that plot were chosen and what you hope the reader would gain from it as well.

Variables

Predictors Airline - some airlines are more expensive than others – how does this impact price of ticket? Date of Journey - flying during certain months is more expensive than others – how does this impact price of ticket? Source - flying out of certain cities is more expensive than others – how does this impact price of ticket? Destination - flying into certain cities is more expensive than others – how does this impact price of ticket? Departure Time - flying at certain times is more expensive than others – how does this impact price of ticket? Total Stops - number of stops may impact the price of the ticket Jet Fuel Price - jet fuel may impact price of ticket ##### Target Price - what we want to predict

Additional Details:

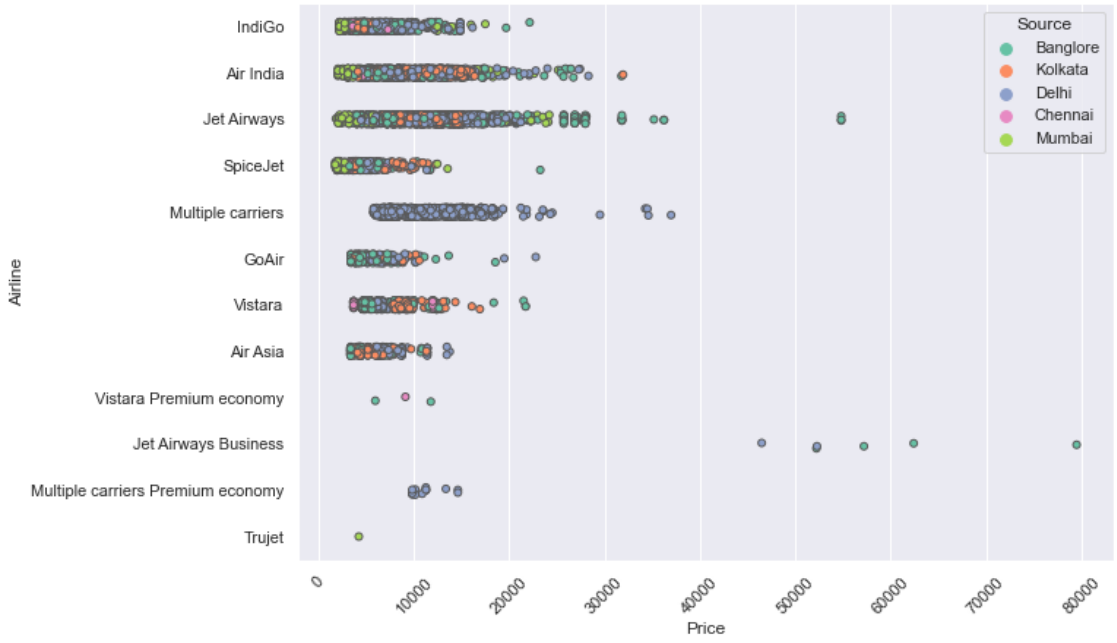
We mostly plot the individual predictors against the target as most of the predictors are categorical and we did not see how they would be correlated with each other (i.e. airline and date of journey)

[24]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

[25]:
```
sns.set(rc = {'figure.figsize':(10,7)})
ax = sns.stripplot(x = "Price", y = "Airline", hue="Source", palette="Set2",␣
 ↪data = merged_data, linewidth = 1)
plt.xticks(rotation=45)
```

[25]:
```
(array([-10000.,      0.,  10000.,  20000.,  30000.,  40000.,  50000.,
         60000.,  70000.,  80000.,  90000.]),
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
```

```
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, '')])
```

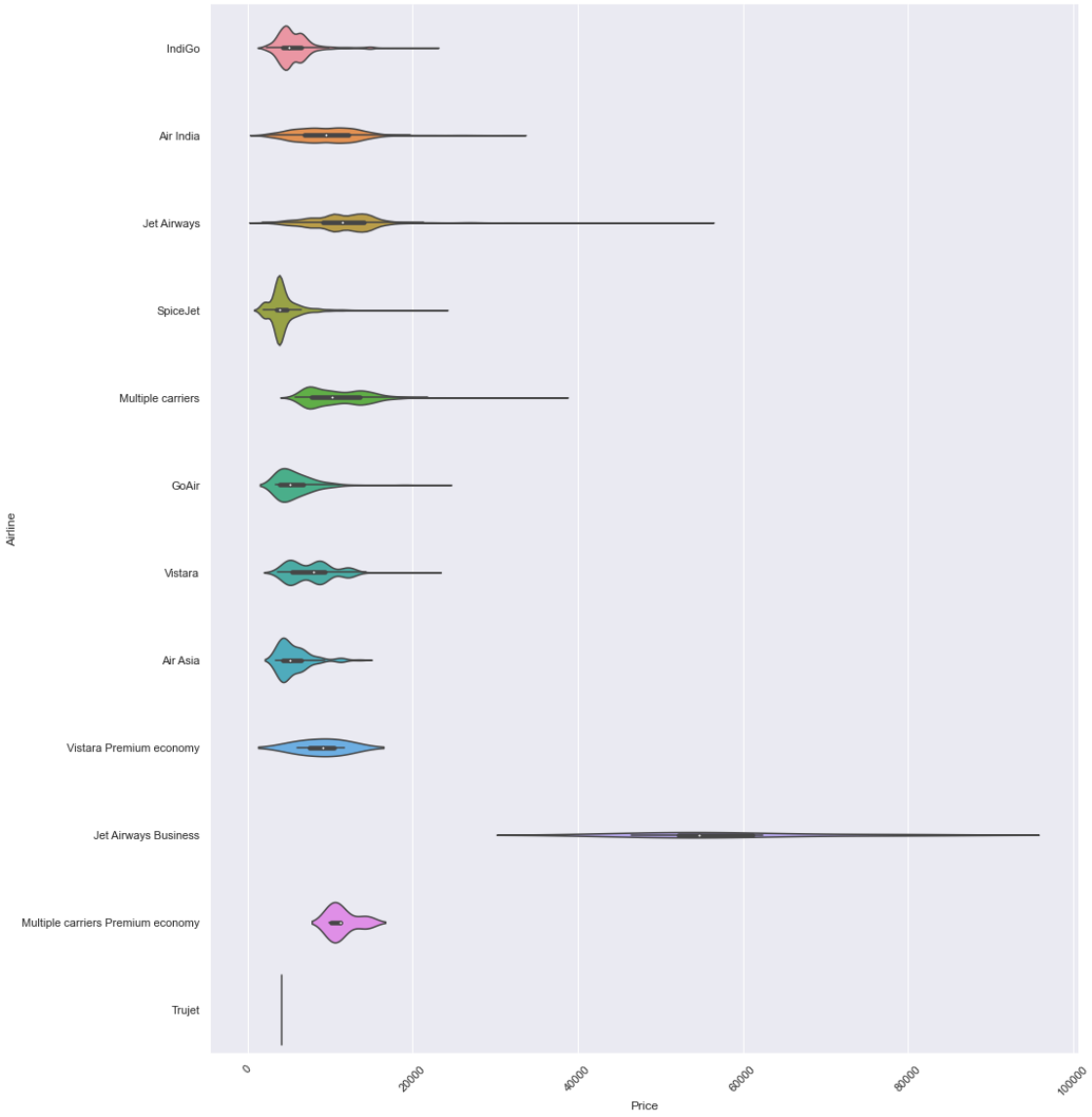


Description of Above Plot In the above plot, we wanted to visualize the relationship between airline and price using a stripplot, as well as incorporate the source location of the flight. As is evident, using a stripplot yields a very busy plot. Therefore, we determined this was not the best mode of visualization.

[26]:
```
sns.set(rc = {'figure.figsize':(10,7)})
sns.catplot(x="Price", y="Airline", kind="violin", data=merged_data, height=15)
plt.xticks(rotation=45)
```

[26]:
```
(array([-20000.,      0.,  20000.,  40000.,  60000.,  80000., 100000.,
        120000.]),
 [Text(-20000.0, 0, '-20000'),
  Text(0.0, 0, '0'),
  Text(20000.0, 0, '20000'),
  Text(40000.0, 0, '40000'),
  Text(60000.0, 0, '60000'),
  Text(80000.0, 0, '80000'),
  Text(100000.0, 0, '100000'),
```

```
Text(120000.0, 0, '120000')])
```



Description of Above Plot Using a violin plot to visualize the relationship between price and airlines is much more clear. From this, we can see where the majority of price data points for an airline lies, as well as how far the outliers reach. From the above plot, we can see that Jet Airways Business is very clearly the most expensive airline of the group.

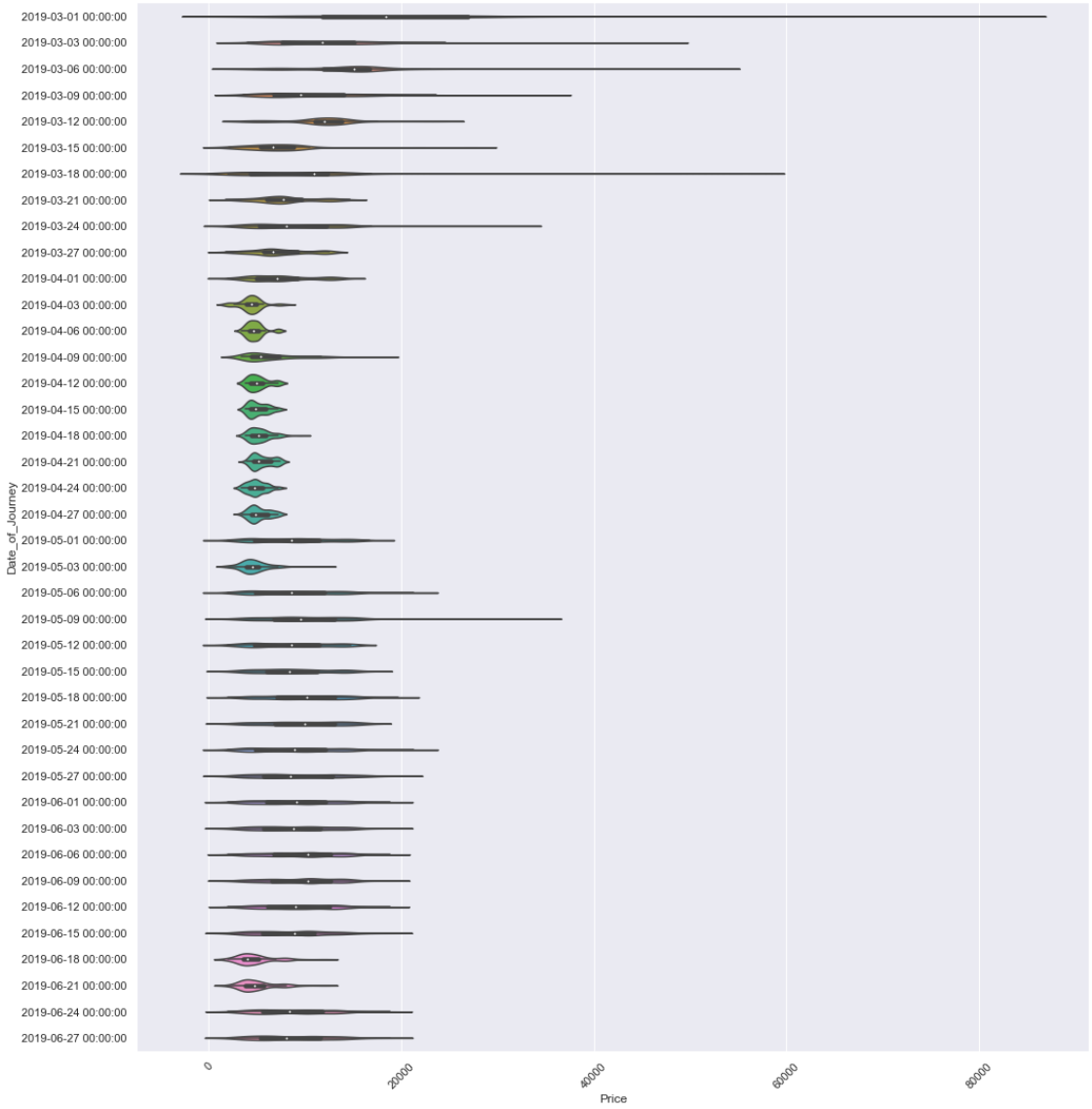
[27]:
```
# convert date of journey strings to datetime objects
from datetime import datetime

for index, date in enumerate(merged_data["Date_of_Journey"]):
```

```
    merged_data["Date_of_Journey"].iloc[index] = datetime.strptime(date, '%d/%m/
↪%Y')
```

[28]:
```
sns.set(rc = {'figure.figsize':(10,7)})
merged_data = merged_data.sort_values(by=['Date_of_Journey'])
sns.catplot(x="Price", y="Date_of_Journey", kind="violin", data=merged_data,␣
↪height=15)
plt.xticks(rotation=45)
```

[28]:
```
(array([-20000.,      0.,  20000.,  40000.,  60000.,  80000., 100000.]),
 [Text(-20000.0, 0, '-20000'),
  Text(0.0, 0, '0'),
  Text(20000.0, 0, '20000'),
  Text(40000.0, 0, '40000'),
  Text(60000.0, 0, '60000'),
  Text(80000.0, 0, '80000'),
  Text(100000.0, 0, '100000')])
```



Description of Above Plot The above plot is also very busy but we think provides a valuable insight into relationship between date of flight and ticket price. The dates have been sorted chronologically and, therefore, we can see that flights in the March timeframe were the most expensive. Many university and high school spring breaks occur during March, which can contribute to a lot of international travel.

[29]:
```
sns.set(rc = {'figure.figsize':(10,7)})
sns.lineplot(x="Date_of_Journey", y="Price", data=merged_data)
plt.xticks(rotation=45)
```

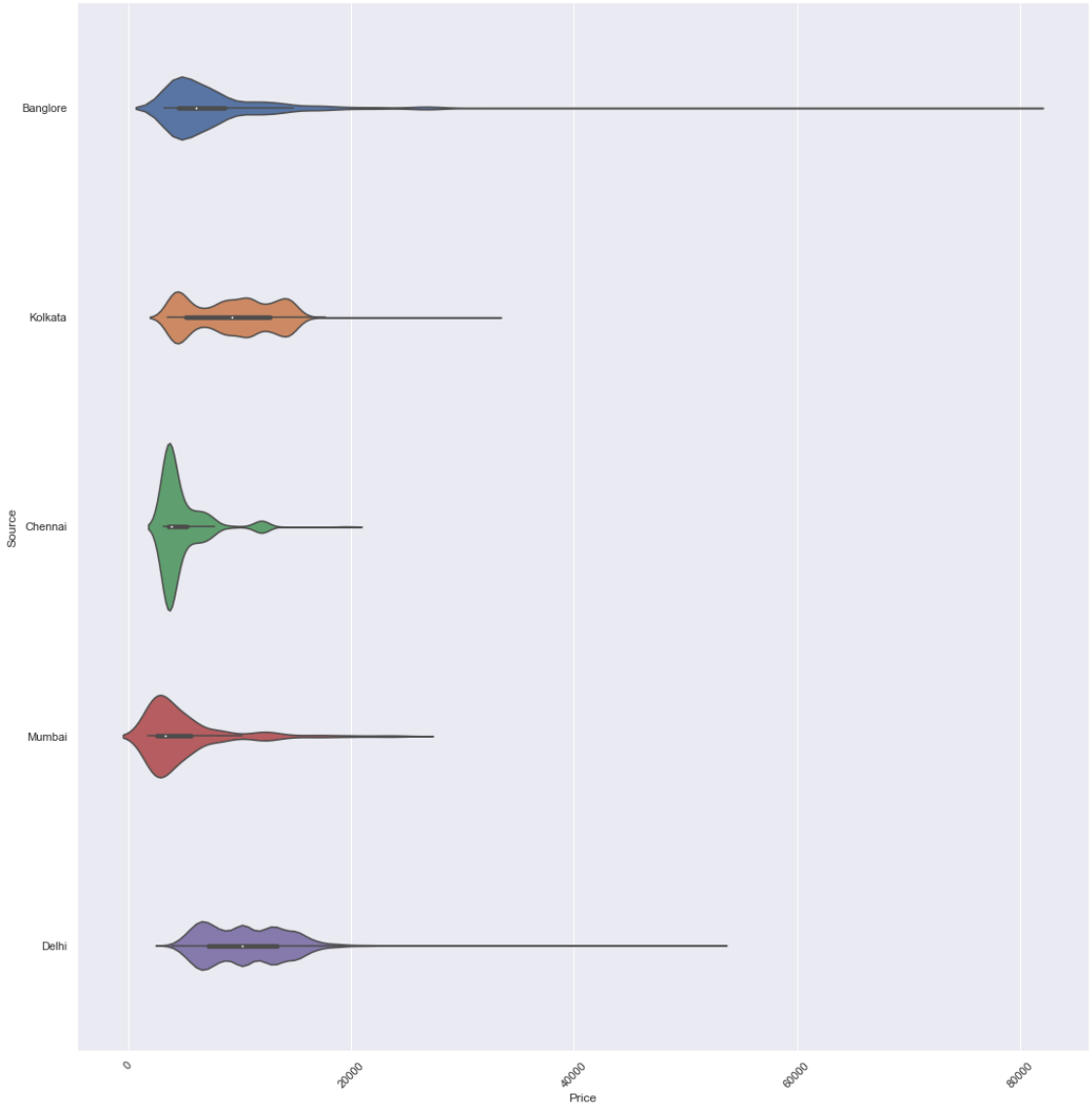
[29]:
```
(array([17956., 17970., 17987., 18001., 18017., 18031., 18048., 18062.,
        18078.]),
```

```
[Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, '')])
```

Description of Above Plot The above plot visualizes the same information as the one above it, just in a different format – a line graph. Again, we clearly see that the beginning of March are the most expensive flights, however, we can also see that the price picks back up again during the June month – the beginning of summer – when people are beginning to travel for summer vacation, potentially.

[30]:
```
sns.set(rc = {'figure.figsize':(10,7)})
sns.catplot(x="Price", y="Source", kind="violin", data=merged_data, height=15)
plt.xticks(rotation=45)
```

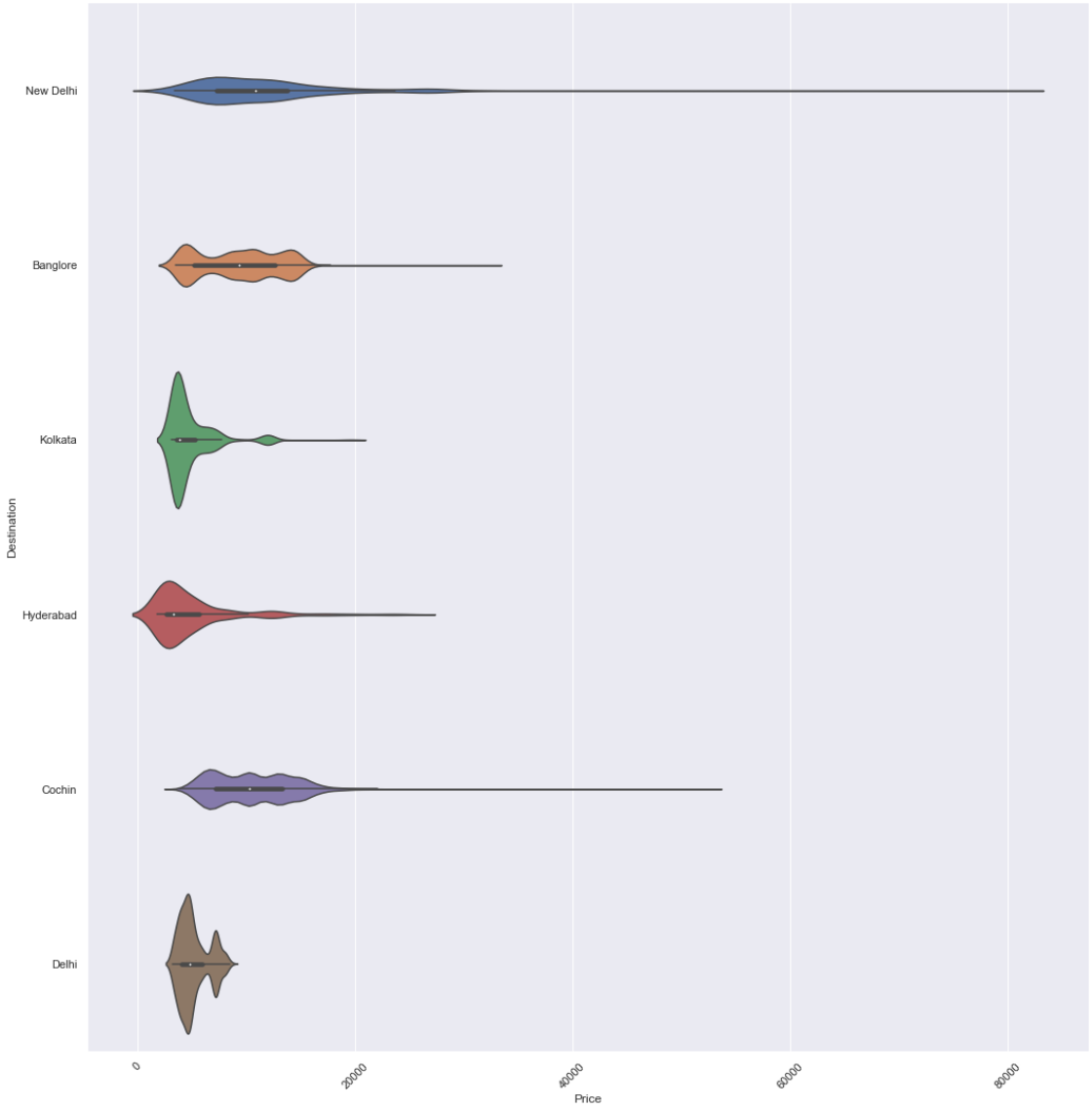
```
[30]: (array([-20000.,      0.,  20000.,  40000.,  60000.,  80000., 100000.]),
       [Text(-20000.0, 0, '-20000'),
        Text(0.0, 0, '0'),
        Text(20000.0, 0, '20000'),
        Text(40000.0, 0, '40000'),
        Text(60000.0, 0, '60000'),
        Text(80000.0, 0, '80000'),
        Text(100000.0, 0, '100000')])
```



Description of Above Plot The above plot visualizes the relationship between the flight source location and the airfare price. As is seen, none of the sources outprice the others by very much, although Banglore does seem to have a very expensive outlier.

[31]:
```
sns.set(rc = {'figure.figsize':(10,7)})
sns.catplot(x="Price", y="Destination", kind="violin", data=merged_data,␣
 ↪height=15)
plt.xticks(rotation=45)
```

[31]:
```
(array([-20000.,      0.,  20000.,  40000.,  60000.,  80000., 100000.]),
 [Text(-20000.0, 0, '-20000'),
  Text(0.0, 0, '0'),
  Text(20000.0, 0, '20000'),
  Text(40000.0, 0, '40000'),
  Text(60000.0, 0, '60000'),
  Text(80000.0, 0, '80000'),
  Text(100000.0, 0, '100000')])
```



Description of Above Plot The above plot explores the relationship between the flight destination location and the price of the airfare. Again, there isn't a lot of variance of prices between the different locations, although New Delhi is evidently the more expensive option to fly into.

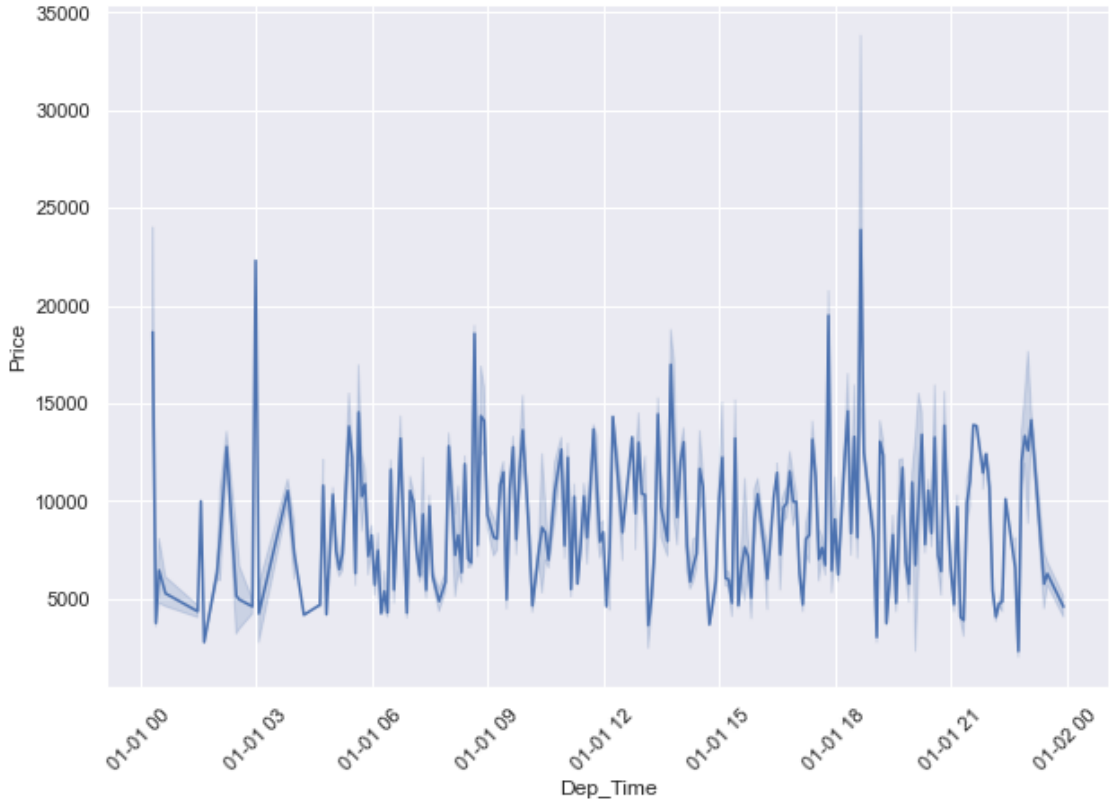
[32]:
```
for index, time in enumerate(merged_data["Dep_Time"]):
    merged_data["Dep_Time"].iloc[index] = datetime.strptime(time,'%H:%M')

merged_data = merged_data.sort_values(by=['Dep_Time'])
```

[33]:
```
from datetime import time

sns.set(rc = {'figure.figsize':(10,7)})
sns.lineplot(x="Dep_Time", y="Price", data=merged_data)
plt.xticks(rotation=45)
```

[33]:
```
(array([-25567.   , -25566.875, -25566.75 , -25566.625, -25566.5  ,
        -25566.375, -25566.25 , -25566.125, -25566.   ]),
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```

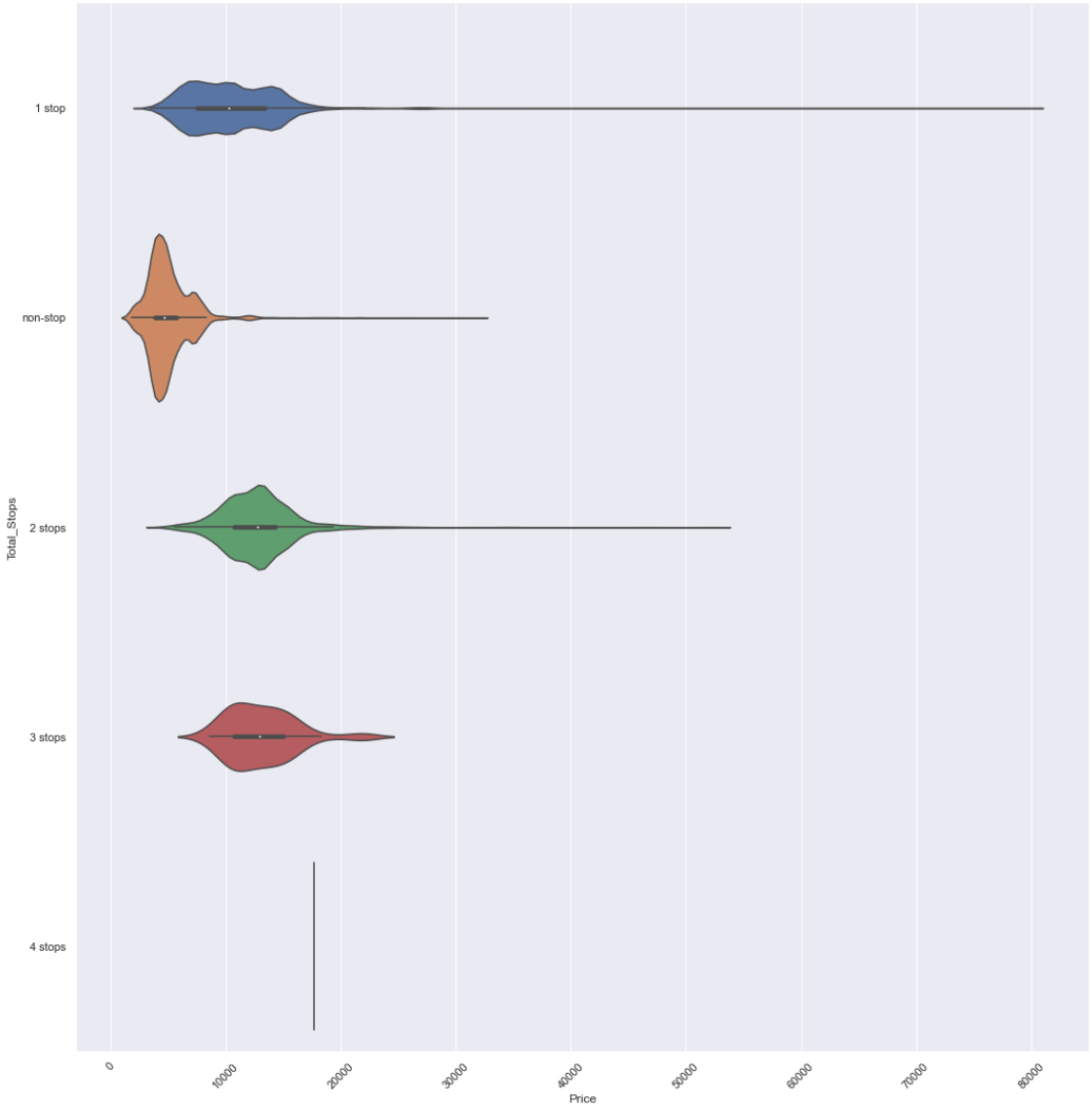


Description of Above Plot The above plot visualizes the price of airfare at certain times during the day over various dates. The end of the x-axis tick is the hour (i.e. 03 in 01-01 03 is 3am). The price seems to peak very early in the morning and then again in the evening, although not in an extreme manner. The rest of the times are very much back and forth, with no obvious pattern being shown in the plot. This demonstrates some relationship between departure time and airfare price.

[34]:
```
sns.set(rc = {'figure.figsize':(10,7)})
sns.catplot(x="Price", y="Total_Stops", kind="violin", data=merged_data,␣
 ↪height=15)
plt.xticks(rotation=45)
```

[34]:
```
(array([-10000.,      0.,  10000.,  20000.,  30000.,  40000.,  50000.,
         60000.,  70000.,  80000.,  90000.]),
 [Text(-10000.0, 0, '-10000'),
  Text(0.0, 0, '0'),
  Text(10000.0, 0, '10000'),
  Text(20000.0, 0, '20000'),
  Text(30000.0, 0, '30000'),
  Text(40000.0, 0, '40000'),
  Text(50000.0, 0, '50000'),
```

```
Text(60000.0, 0, '60000'),
Text(70000.0, 0, '70000'),
Text(80000.0, 0, '80000'),
Text(90000.0, 0, '90000')])
```

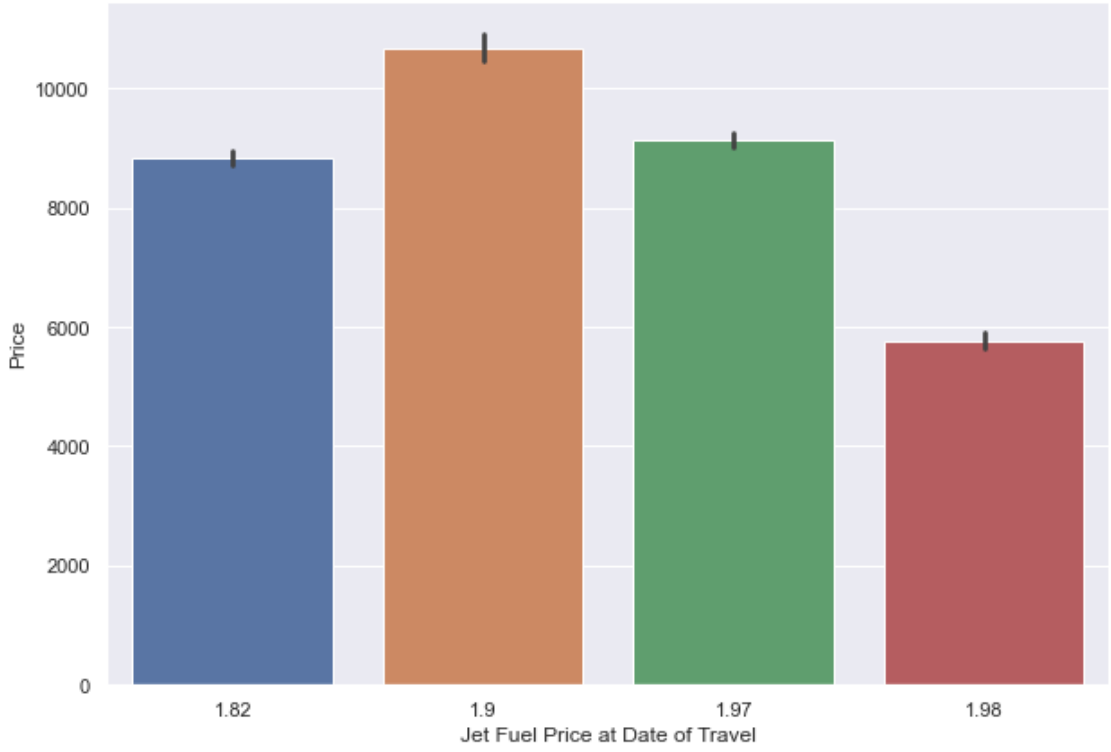


Description of Above Plot The above plot visuzalizes the relationship between the amount of stops during the entire flight and the price of the airfare. From this, we can see that non-stop is the cheapest option, which more stops slightly increasing the overall price of the airfare.

[35]:
```
merged_data["Jet Fuel Price at Date of Travel"] = merged_data["Jet Fuel Price␣
↪at Date of Travel"].astype('float')
sns.set(rc = {'figure.figsize':(10,7)})
```

```
sns.barplot(data=merged_data,x="Jet Fuel Price at Date of Travel",y="Price")
```

[35]:
```
<AxesSubplot:xlabel='Jet Fuel Price at Date of Travel', ylabel='Price'>
```

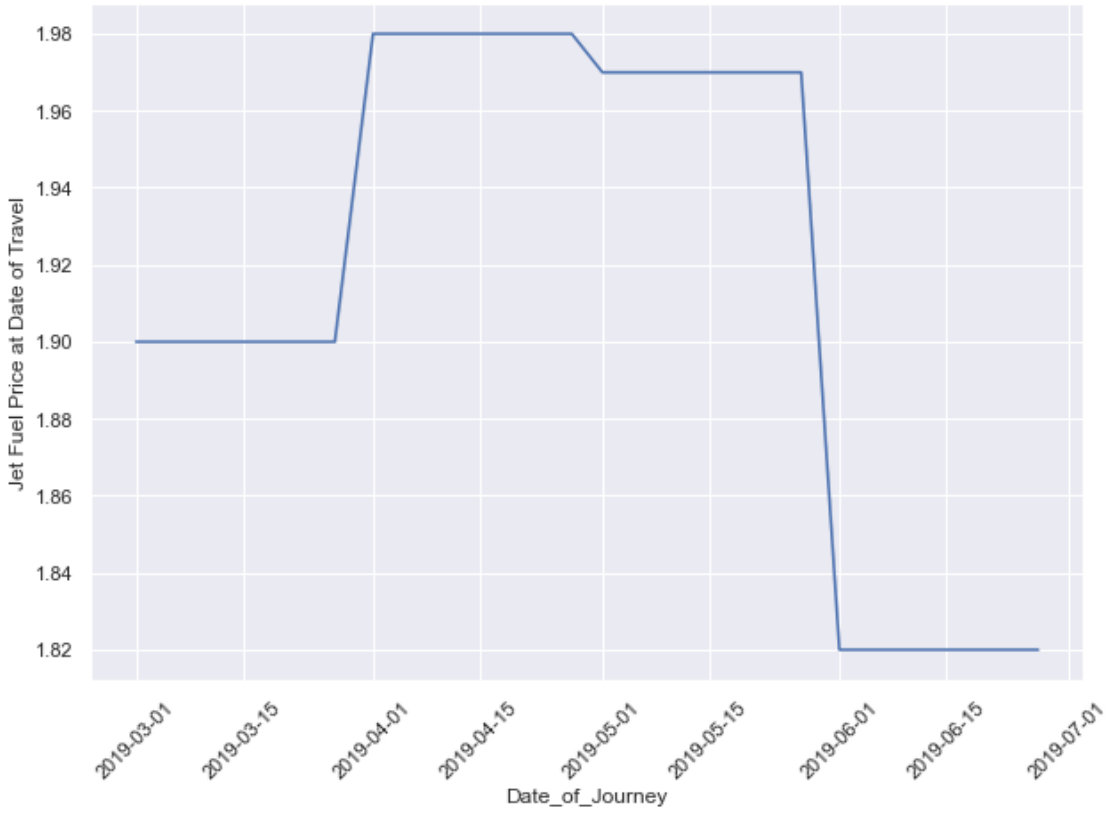


Description of Above Plot The above plot visualizes the relationship between jet fuel price and airfare price. This is interesting because, it seems when jet fuel is the most expensive, the airfare is cheapest. This indicates that there is not a linear relationship between the two variables.

[36]:
```
sns.set(rc = {'figure.figsize':(10,7)})
merged_data = merged_data.sort_values(by=['Date_of_Journey'])
sns.lineplot(x="Date_of_Journey", y="Jet Fuel Price at Date of Travel",␣
 ↪data=merged_data)
plt.xticks(rotation=45)
```

[36]:
```
(array([17956., 17970., 17987., 18001., 18017., 18031., 18048., 18062.,
        18078.]),
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
```

```
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, '')])
```



Description of Above Plot The above plot visualizes the jet fuel price by date. When we refer to the lineplot that visualizes date of journey against price (put below again for convenience), we see that the price of the airfare is actually the least expensive when jet fuel is the most expensive. This indicates that there is little (or at least not a linear) relationship between jet fuel and airfare price.

[37]:
```
sns.set(rc = {'figure.figsize':(10,7)})
sns.lineplot(x="Date_of_Journey", y="Price", data=merged_data)
plt.xticks(rotation=45)
```

[37]:
```
(array([17956., 17970., 17987., 18001., 18017., 18031., 18048., 18062.,
        18078.]),
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
```

```
Text(0, 0, ''),
Text(0, 0, ''),
Text(0, 0, ''),
Text(0, 0, '')])
```