Project 2

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.

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.

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.

In [1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```python
import seaborn as sns
import nbconvert
import datetime
```

Original Dataset

In [2]:

```python
# !kaggle datasets download -d zwartfreak/airline-fare-prediction
```

In [3]:

```python
flights_df = pd.read_excel('flights.xlsx')
flights_df.head()
```

Out[3]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Sto
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-sto
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 sto
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 sto
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 sto
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 sto

2nd Dataset

In [4]:

```python
# !kaggle datasets download -d mabusalah/brent-oil-prices
```

In [5]:

```python
oil_df = pd.read_csv('BrentOilPrices.csv')
oil_df.head()
```

Out[5]:

	Date	Price
0	20-May-87	18.63
1	21-May-87	18.45
2	22-May-87	18.55
3	25-May-87	18.60

	Date	Price
4	26-May-87	18.63

3rd Dataset

In [6]:

```
# !kaggle datasets download -d sudalairajkumar/daily-temperature-of-major-cities
```

In [7]:

```
temp_df = pd.read_csv('city_temperature.csv')
temp_df.head()
```

```
/home/sam/anaconda3/envs/aplML/lib/python3.9/site-packages/IPython/core/interactiveshel
l.py:3457: DtypeWarning: Columns (2) have mixed types.Specify dtype option on import or
set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Out[7]:

	Region	Country	State	City	Month	Day	Year	AvgTemperature
0	Africa	Algeria	NaN	Algiers	1	1	1995	64.2
1	Africa	Algeria	NaN	Algiers	1	2	1995	49.4
2	Africa	Algeria	NaN	Algiers	1	3	1995	48.8
3	Africa	Algeria	NaN	Algiers	1	4	1995	46.4
4	Africa	Algeria	NaN	Algiers	1	5	1995	47.9

Datasets Description

The Airfare Prediction Dataset contains general flight data (airline, source, destination, duration, ..., and **price**). The dataset is set up for flight price prediction. It contains interesting data from different airlines for a few pairs of (source, destinations) over 2019. This also makes it interesting for investigating the difference between airlines pricing models and what aspects of the flight have larger contributions.

When thinking about contributing factors to airfare, the first thing comes to mind is oil prices. I found the Brent Oil Prices dataset , it has the daily historical prices and it is collected from the U.S. Energy Information Administration. It has data from 17th of May 1987 until the 25th of February 2020, but I am only interested in the data from 2019

For the third dataset, I chose to explore if changes in temperature correlate with the pricing of flights. Daily Temperatures of Major Cities contains daily historical temperature data for tens of major cities around the globe.

Questions:
What is the level of correlation between oil prices and airfare? Is temperature a predictor of flight price? How strong? Which data features strongly relate to the flight price? Can the combined dataset perform better than the originl on the prediction problem?

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.

Flight Data

In [8]:
```
flights_df.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
```

In [9]:
```
flights_df.isnull().sum()
```

Out[9]:
```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              1
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        1
Additional Info    0
Price              0
dtype: int64
```

In [10]:
```
flights_df.dropna(inplace=True)
```

In [11]:
```
flights_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10682 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          10682 non-null  object
 1   Date_of_Journey  10682 non-null  object
 2   Source           10682 non-null  object
 3   Destination      10682 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10682 non-null  object
 6   Arrival_Time     10682 non-null  object
 7   Duration         10682 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10682 non-null  object
 10  Price            10682 non-null  int64
dtypes: int64(1), object(10)
memory usage: 1001.4+ KB
```

So one row was dropped.

The printout also shows type of each column. Date_of_Journey, Dep_Time, Arrival_Time should not be object type, more on that later.

In [12]:
```
# flights_df['Date_of_Journey'] = [datetime.datetime.strptime(x,'%d/%m/%Y').strftime('%d-%m-%Y') for x in flights_df['Date_of_Journey']]
# flights_df['Date_of_Journey']
```

In [13]:
```
flights_df['Date_of_Journey'] = pd.to_datetime(flights_df['Date_of_Journey'], format='%d/%m/%Y')
flight_dates = flights_df['Date_of_Journey'].dt.strftime('%d-%m-%Y')
```

In [14]:
```
flights_df['Arrival_Time'] = pd.to_datetime(flights_df['Arrival_Time'])
flights_df['Dep_Time'] = pd.to_datetime(flights_df['Dep_Time'])
```

In [15]:
```
flights_df.dtypes
```

Out[15]:
```
Airline                    object
Date_of_Journey    datetime64[ns]
Source                     object
Destination                object
Route                      object
Dep_Time           datetime64[ns]
Arrival_Time       datetime64[ns]
Duration                   object
Total_Stops                object
Additional_Info            object
Price                       int64
dtype: object
```

In [16]:
```python
sorted_dates = sorted(flights_df['Date_of_Journey'])
print('flights time boundries: {} - {}'.format(sorted_dates[0], sorted_dates[-1]))
```

```
flights time boundries: 2019-03-01 00:00:00 - 2019-06-27 00:00:00
```

In [17]:
```python
flights_df['Dep_Time_h'] = flights_df['Dep_Time'].dt.hour
flights_df['Dep_Time_m'] = flights_df['Dep_Time'].dt.minute
```

In [18]:
```python
flights_df['Arv_Time_h'] = flights_df['Arrival_Time'].dt.hour
flights_df['Arv_Time_m'] = flights_df['Arrival_Time'].dt.minute
```

In [19]:
```python
Dur_in_mins = []
for dur in flights_df['Duration']:
    l = dur.split(' ')
    mins = 0
    if len(l) == 2:
        mins = int(l[0][0:-1]) * 60 + int(l[1][0:-1])
    else:
        if l[0][-1] == 'h':
            mins = int(l[0][0:-1]) * 60
        else:
            mins = int(l[0][0:-1])

    Dur_in_mins.append(mins)
flights_df['Duration_mins'] = Dur_in_mins
```

In [20]:
```python
flights_df.drop(columns=['Duration', 'Arrival_Time', 'Dep_Time'], inplace = True)
```

In [21]:
```python
flights_df.head()
```

Out[21]:

	Airline	Date_of_Journey	Source	Destination	Route	Total_Stops	Additional Info	Price	Dep_Time_h
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	non-stop	No info	3898	22
1	Air India	2019-05-01	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7663	5
2	Jet Airways	2019-06-09	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13883	9

	Airline	Date_of_Journey	Source	Destination	Route	Total_Stops	Additional Info	Price	Dep_Time_h
3	IndiGo	2019-05-12	Kolkata	Banglore	CCU → NAG → BLR	1 stop	No info	6219	18
4	IndiGo	2019-03-01	Banglore	New Delhi	BLR → NAG → DEL	1 stop	No info	13303	16

In [22]:
```
flights_df.dtypes
```

Out[22]:
```
Airline                      object
Date_of_Journey      datetime64[ns]
Source                       object
Destination                  object
Route                        object
Total_Stops                  object
Additional Info              object
Price                         int64
Dep_Time_h                    int64
Dep_Time_m                    int64
Arv_Time_h                    int64
Arv_Time_m                    int64
Duration_mins                 int64
dtype: object
```

Oil Data

In [23]:
```
oil_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8554 entries, 0 to 8553
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    8554 non-null   object 
 1   Price   8554 non-null   float64
dtypes: float64(1), object(1)
memory usage: 133.8+ KB
```

In [24]:
```
oil_df.loc[list(np.random.choice(oil_df.shape[0], 5))]
```

Out[24]:

	Date	Price
6388	24-Jul-12	103.57
1443	8-Jan-93	17.23
3010	29-Mar-99	14.34
8129	31-May-19	66.78

	Date	Price
7951	14-Sep-18	77.87

In [25]:

```
clean_dates = []
dates_to_drop = []
processed = 0
for i,date in enumerate(oil_df['Date']):
    try:
        d = datetime.datetime.strptime(date,'%d-%b-%y').strftime('%d-%m-%Y')
        oil_df['Date'] = oil_df['Date'].replace([date], d)
        processed += 1
    except:
        dates_to_drop.append(i)
        pass

processed
```

Out[25]:

```
8360
```

In [26]:

```
oil_df['Date']
```

Out[26]:

```
0         20-05-1987
1         21-05-1987
2         22-05-1987
3         25-05-1987
4         26-05-1987
             ...
8549    Jan 19, 2021
8550    Jan 20, 2021
8551    Jan 21, 2021
8552    Jan 22, 2021
8553    Jan 25, 2021
Name: Date, Length: 8554, dtype: object
```

In [27]:

```
oil_df['Date'][dates_to_drop]
```

Out[27]:

```
8360    Apr 22, 2020
8361    Apr 23, 2020
8362    Apr 24, 2020
8363    Apr 27, 2020
8364    Apr 28, 2020
             ...
8549    Jan 19, 2021
8550    Jan 20, 2021
8551    Jan 21, 2021
8552    Jan 22, 2021
8553    Jan 25, 2021
Name: Date, Length: 194, dtype: object
```

In [28]:
```
oil_df.drop(index=dates_to_drop, inplace=True)
```

In [29]:
```
oil_df['Year'] = pd.to_datetime(oil_df['Date']).dt.year
oil_df['Year'] = oil_df['Year'].astype(str)
```

In [30]:
```
oil_df = oil_df[oil_df['Year'] == '2019']
oil_df.drop(columns=['Year'], inplace=True)
oil_df.shape
```

Out[30]:
```
(258, 2)
```

Temperature Data

In [31]:
```
temp_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2906327 entries, 0 to 2906326
Data columns (total 8 columns):
 #   Column          Dtype  
---  ------          -----  
 0   Region          object 
 1   Country         object 
 2   State           object 
 3   City            object 
 4   Month           int64  
 5   Day             int64  
 6   Year            int64  
 7   AvgTemperature  float64
dtypes: float64(1), int64(3), object(4)
memory usage: 177.4+ MB
```

In [32]:
```
temp_df.isnull().sum()
```

Out[32]:
```
Region                  0
Country                 0
State             1450990
City                    0
Month                   0
Day                     0
Year                    0
AvgTemperature          0
dtype: int64
```

In [33]:
```
temp_df = temp_df[temp_df['Country'] == 'India'].reset_index(drop= True)
temp_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37063 entries, 0 to 37062
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
```

```
---  ------          --------------  -----
 0   Region          37063 non-null  object
 1   Country         37063 non-null  object
 2   State           0 non-null      object
 3   City            37063 non-null  object
 4   Month           37063 non-null  int64
 5   Day             37063 non-null  int64
 6   Year            37063 non-null  int64
 7   AvgTemperature  37063 non-null  float64
dtypes: float64(1), int64(3), object(4)
memory usage: 2.3+ MB
```

In [34]:

```python
temp_df.drop(columns=['State'])
```

Out[34]:

	Region	Country	City	Month	Day	Year	AvgTemperature
0	Asia	India	Bombay (Mumbai)	1	1	1995	71.8
1	Asia	India	Bombay (Mumbai)	1	2	1995	72.0
2	Asia	India	Bombay (Mumbai)	1	3	1995	70.3
3	Asia	India	Bombay (Mumbai)	1	4	1995	69.7
4	Asia	India	Bombay (Mumbai)	1	5	1995	71.3
...	...	...	...	...	...	...	...
37058	Asia	India	Delhi	5	8	2020	89.9
37059	Asia	India	Delhi	5	9	2020	92.3
37060	Asia	India	Delhi	5	10	2020	81.9
37061	Asia	India	Delhi	5	11	2020	84.7
37062	Asia	India	Delhi	5	12	2020	88.1

37063 rows × 7 columns

In [35]:

```python
temp_df = temp_df[temp_df['Year'] == 2019]
print('new length: {}'.format(len(temp_df)) )
temp_df.head()
```

```
new length: 1460
```

Out[35]:

	Region	Country	State	City	Month	Day	Year	AvgTemperature
8767	Asia	India	NaN	Bombay (Mumbai)	1	1	2019	76.7
8768	Asia	India	NaN	Bombay (Mumbai)	1	2	2019	77.9
8769	Asia	India	NaN	Bombay (Mumbai)	1	3	2019	77.8
8770	Asia	India	NaN	Bombay (Mumbai)	1	4	2019	79.3
8771	Asia	India	NaN	Bombay (Mumbai)	1	5	2019	76.7

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.

Flight Dataset

In [36]:

```
flights_df.describe(include='all', datetime_is_numeric=True)
```

Out[36]:

	Airline	Date_of_Journey	Source	Destination	Route	Total_Stops	Additional Info	Price
count	10682	10682	10682	10682	10682	10682	10682	10682.000000
unique	12	NaN	5	6	128	5	10	NaN
top	Jet Airways	NaN	Delhi	Cochin	DEL → BOM → COK	1 stop	No info	NaN
freq	3849	NaN	4536	4536	2376	5625	8344	NaN
mean	NaN	2019-05-04 19:56:32.398427392	NaN	NaN	NaN	NaN	NaN	9088.214567
min	NaN	2019-03-01 00:00:00	NaN	NaN	NaN	NaN	NaN	1760.000000
25%	NaN	2019-03-27 00:00:00	NaN	NaN	NaN	NaN	NaN	5278.000000
50%	NaN	2019-05-15 00:00:00	NaN	NaN	NaN	NaN	NaN	8373.000000
75%	NaN	2019-06-06 00:00:00	NaN	NaN	NaN	NaN	NaN	12374.000000
max	NaN	2019-06-27 00:00:00	NaN	NaN	NaN	NaN	NaN	79513.000000
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4611.548810

In [37]:

```
num_cols = flights_df._get_numeric_data().columns
flights_num = flights_df[num_cols]
flights_cat = flights_df.drop(columns=num_cols)
print('numerical columns: {}\n'.format(flights_num.columns.to_list()))
print('catagorical columns: {}'.format(flights_cat.columns.to_list()))
```

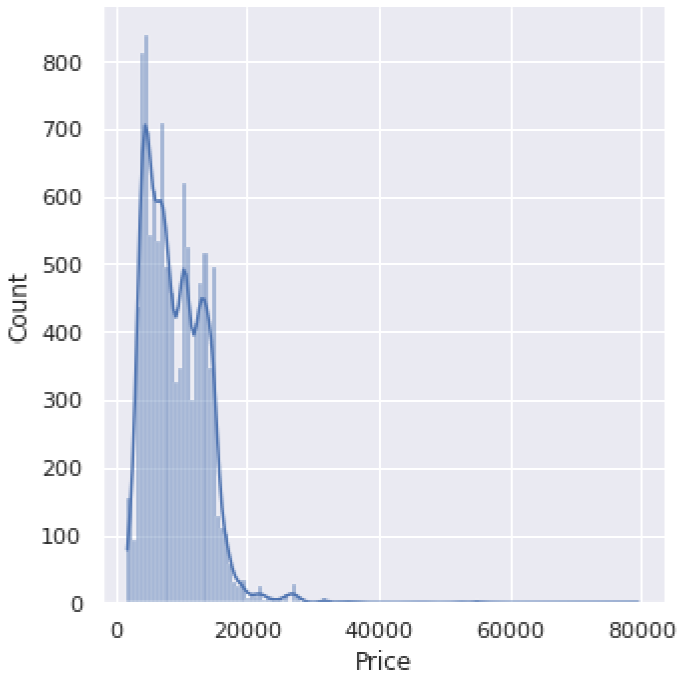
```
numerical columns: ['Price', 'Dep_Time_h', 'Dep_Time_m', 'Arv_Time_h', 'Arv_Time_m', 'Du
ration_mins']

catagorical columns: ['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route', 'T
otal_Stops', 'Additional Info']
```

Explore data

We can start exploring the price relation to other variable

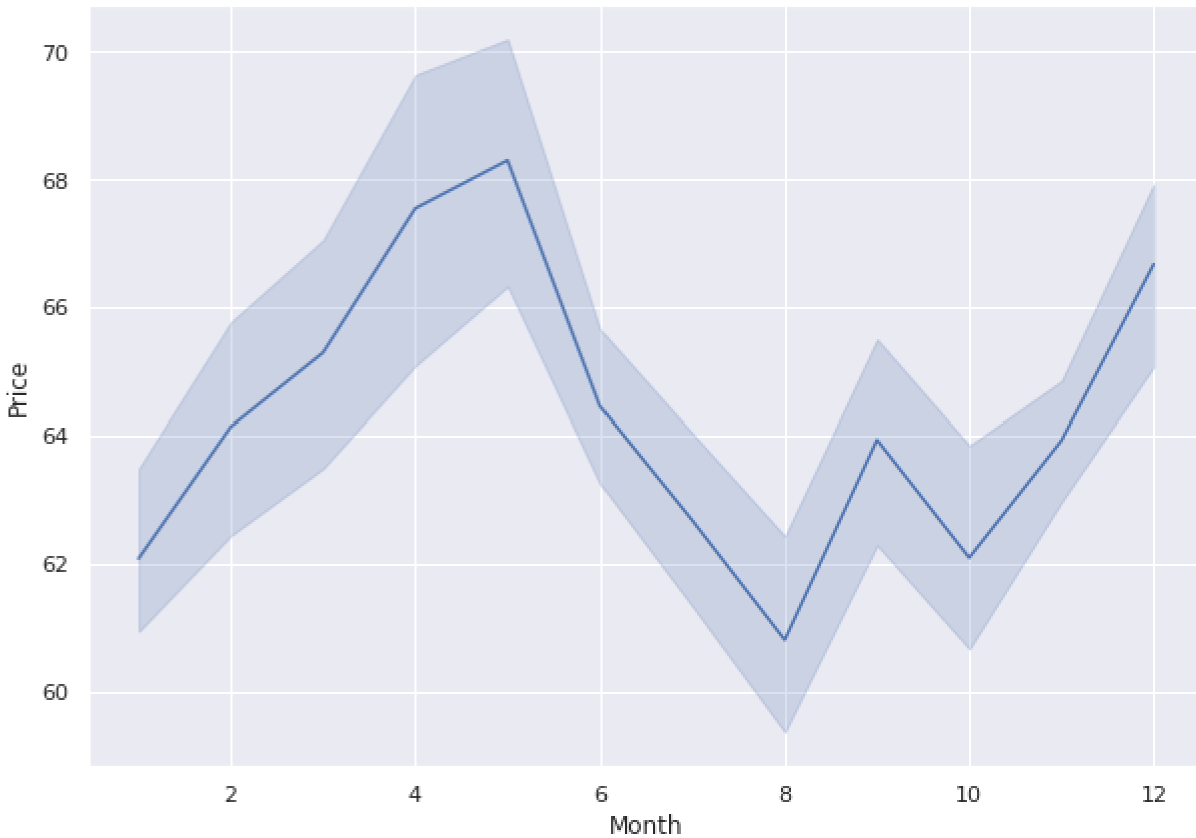
In [38]:
```
sns.set_theme()
sns.set(rc = {'figure.figsize':(10,7)})
sns.displot(flights_df['Price'], kde=True)
```

Out[38]:
```
<seaborn.axisgrid.FacetGrid at 0x7ff9615e2040>
```



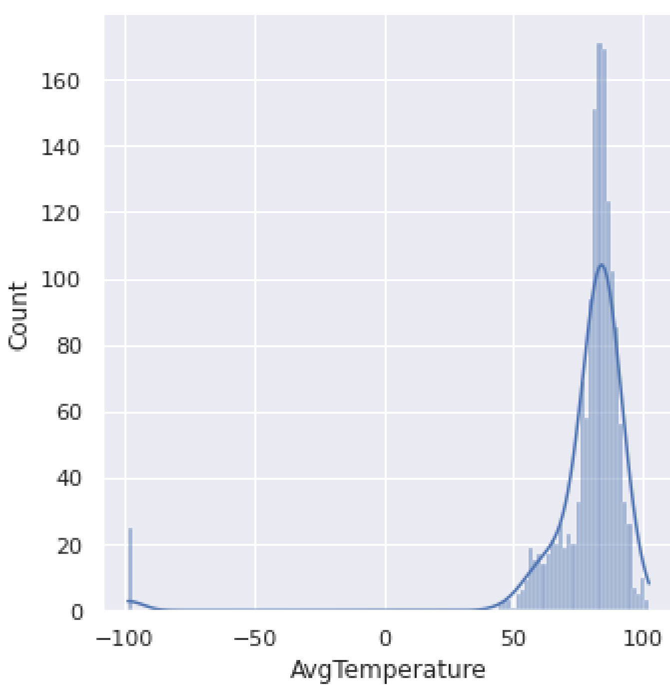
In [39]:
```
oil_df['Month'] = pd.to_datetime(oil_df['Date']).dt.month
sns.lineplot(x='Month', y='Price', data = oil_df)
```

Out[39]:
```
<AxesSubplot:xlabel='Month', ylabel='Price'>
```



In [40]:
```
sns.displot(temp_df['AvgTemperature'], kde=True)
```

Out[40]:
```
<seaborn.axisgrid.FacetGrid at 0x7ff95e3cd160>
```

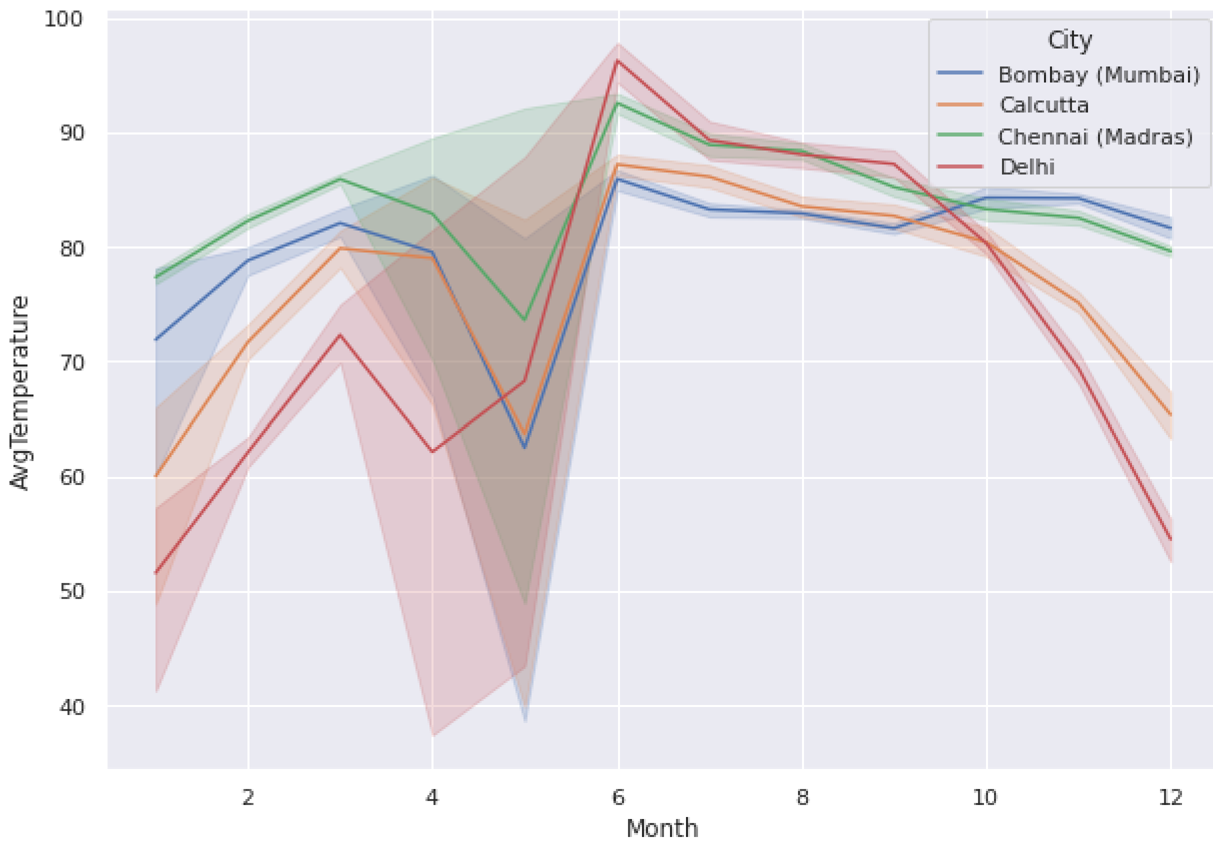


In [41]:
```
temp_df['Date'] = temp_df[['Day', 'Month', 'Year']].apply(lambda row: '-
'.join(row.values.astype(str)), axis=1)
sns.lineplot(data=temp_df[['Month',
                           'AvgTemperature',
```

```
                                'City']],
                        x='Month',y='AvgTemperature', hue='City')
```

Out[41]:

```
<AxesSubplot:xlabel='Month', ylabel='AvgTemperature'>
```



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).

Merging flights & oil prices

In [42]:

```
oil_df['Date']
```

Out[42]:

```
8024    02-01-2019
8025    03-01-2019
8026    04-01-2019
8027    07-01-2019
8028    08-01-2019
           ...
8277    25-12-2019
8278    26-12-2019
8279    27-12-2019
8280    30-12-2019
```

```
8281    31-12-2019
Name: Date, Length: 258, dtype: object
```

In [43]:

```python
shared_dates = list(set(oil_df['Date']) & set(flight_dates))
print("Found {} shared dates out of a total of {} unique 
dates".format(len(shared_dates) ,len(set(flight_dates))) )
```

```
Found 28 shared dates out of a total of 40 unique dates
```

We found the 28 shared dates between the two datasets, out of 40 unique dates. We will copy the gas price for the shared indices and interpolate the missing values.

In [44]:

```python
gas_price = []
for date in flight_dates:
    if date in shared_dates:
        gas_price.append(oil_df[oil_df['Date']==date]['Price'].values[0])
    else:
        gas_price.append( np.mean(oil_df['Price']) )
len(gas_price)
```

Out[44]:

```
10682
```

In [45]:

```python
combined_df = flights_df.copy(deep = True)

combined_df['Gas_Price'] = gas_price
```

In [46]:

```python
combined_df.head()
```

Out[46]:

	Airline	Date_of_Journey	Source	Destination	Route	Total_Stops	Additional Info	Price	Dep_Time_h
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	non-stop	No info	3898	22
1	Air India	2019-05-01	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7663	5
2	Jet Airways	2019-06-09	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13883	9
3	IndiGo	2019-05-12	Kolkata	Banglore	CCU → NAG → BLR	1 stop	No info	6219	18
4	IndiGo	2019-03-01	Banglore	New Delhi	BLR → NAG → DEL	1 stop	No info	13303	16

Merging (flight + oil prices) & temperature data

In [47]:

```
temp_df['City'].replace(list(['Bombay (Mumbai)', 'Calcutta', 'Chennai (Madras)']),
['Mumbai', 'KolKata','Chennai'], inplace=True)
print('new city names: {}'.format(temp_df['City'].unique()))

temp_cities = list(temp_df['City'].unique())
temp_df['Date'] = pd.to_datetime(temp_df['Date'], format='%d-%m-%Y').dt.strftime('%d-
%m-%Y')
```

```
new city names: ['Mumbai' 'KolKata' 'Chennai' 'Delhi']
```

In [48]:

```
src_temp = []
dest_temp = []

temp_shared_dates = list(set(temp_df['Date']) & set(flight_dates))

for idx, flight in flights_df.iterrows():

    if flight_dates[idx] in temp_shared_dates:

        temps_at_date = temp_df[ temp_df['Date'] == flight_dates[idx] ]
        temperature = 0

        if flight['Source'] in temp_cities:
            row = temps_at_date[ temps_at_date['City'] == flight['Source'] ]
            temperature = row['AvgTemperature'].to_numpy()[0]
        else:
            temperature = np.mean( temps_at_date['AvgTemperature'].values )

        src_temp.append(temperature)

        if flight['Destination'] in temp_cities:
            row = temps_at_date[ temps_at_date['City'] == flight['Destination'] ]
            temperature = row['AvgTemperature'].to_numpy()[0]

        else:
            temperature = np.mean( temps_at_date['AvgTemperature'].values )

        dest_temp.append(temperature)
```

In [49]:

```
print('{} source temperature values\n{} destination temperature
values\n'.format(len(src_temp), len(dest_temp)))
```

```
print('some source temperatures {}\n'.format(src_temp[:3]))
print('some destination temperatures {}\n'.format(dest_temp[:3]))
```

```
10682 source temperature values
10682 destination temperature values

some source temperatures [83.42500000000001, 89.775, 99.4]

some destination temperatures [83.42500000000001, 89.775, 93.275]
```

And now merging our extracted data points to the combined DataFrame.

In [50]:
```
flights_df['Source'].unique()
```

Out[50]:
```
array(['Banglore', 'Kolkata', 'Delhi', 'Chennai', 'Mumbai'], dtype=object)
```

In [51]:
```
combined_df['Src_Temperature'] = src_temp
combined_df['Dest_Temperature'] = dest_temp
combined_df.head()
```

Out[51]:

	Airline	Date_of_Journey	Source	Destination	Route	Total_Stops	Additional Info	Price	Dep_Time_h
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	non-stop	No info	3898	22
1	Air India	2019-05-01	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7663	5
2	Jet Airways	2019-06-09	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13883	9
3	IndiGo	2019-05-12	Kolkata	Banglore	CCU → NAG → BLR	1 stop	No info	6219	18
4	IndiGo	2019-03-01	Banglore	New Delhi	BLR → NAG → DEL	1 stop	No info	13303	16

We have the full combined dataset. Let's check if everything looks alright.

In [52]:
```
combined_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10682 entries, 0 to 10682
Data columns (total 16 columns):
```

```
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Airline            10682 non-null  object
 1   Date_of_Journey    10682 non-null  datetime64[ns]
 2   Source             10682 non-null  object
 3   Destination        10682 non-null  object
 4   Route              10682 non-null  object
 5   Total_Stops        10682 non-null  object
 6   Additional Info    10682 non-null  object
 7   Price              10682 non-null  int64
 8   Dep_Time_h         10682 non-null  int64
 9   Dep_Time_m         10682 non-null  int64
 10  Arv_Time_h         10682 non-null  int64
 11  Arv_Time_m         10682 non-null  int64
 12  Duration_mins      10682 non-null  int64
 13  Gas_Price          10682 non-null  float64
 14  Src_Temperature    10682 non-null  float64
 15  Dest_Temperature   10682 non-null  float64
dtypes: datetime64[ns](1), float64(3), int64(6), object(6)
memory usage: 1.6+ MB
```

Everything looks good. The dataset doesn't contain any null values and the variable types are set correctly.

Let's go over the variables again before I choose a subset to move forward with to the next part.

Catagorical Variables

In [53]:
```
combined_df.describe(include = 'object')
```

Out[53]:

	Airline	Source	Destination	Route	Total_Stops	Additional Info
count	10682	10682	10682	10682	10682	10682
unique	12	5	6	128	5	10
top	Jet Airways	Delhi	Cochin	DEL → BOM → COK	1 stop	No info
freq	3849	4536	4536	2376	5625	8344

x	variable	dtype	description	Part5?	Why
1	Airline	object	airline name	☑	It is save to assume the airline name could influence the price
2	Source	object	source city	☑	Some cities have cheaper/more expensive tickets based on demand, resource, and other factors
3	Destination	object	destination city	☑	//
4	Route	object	list of stops, if any		There are a lot of different unique routes (128). I don't believe the specific stop codes have significance
5	Total_Stops	object	number of stops	☑	The number of stops could definitely influence the flight price
6	Add. Info	object	random info (mostly "No Info"		Mostly useless data

Numerical Variables

In [54]:
```
combined_df.describe(include = ['int64','float64'])
```

Out[54]:

	Price	Dep_Time_h	Dep_Time_m	Arv_Time_h	Arv_Time_m	Duration_mins	Gas_Pr
count	10682.000000	10682.000000	10682.000000	10682.000000	10682.000000	10682.000000	10682.0000
mean	9088.214567	12.491013	24.409287	13.349186	24.690601	643.020502	66.5594
std	4611.548810	5.748820	18.767801	6.859317	16.506808	507.830133	3.4390
min	1760.000000	0.000000	0.000000	0.000000	0.000000	5.000000	61.6600
25%	5278.000000	8.000000	5.000000	8.000000	10.000000	170.000000	64.3198
50%	8373.000000	11.000000	25.000000	14.000000	25.000000	520.000000	64.5100
75%	12374.000000	18.000000	40.000000	19.000000	35.000000	930.000000	69.0800
max	79513.000000	23.000000	55.000000	23.000000	55.000000	2860.000000	73.5900

x	variable	dtype	description	Part5?	Why
7	Price	int64	Fligt price, target variable	☑	*Target*
8	Dep_Time_h	int64	The hour of departure	☑	departure hour can have an effect(ex.: early flights could be cheaper)
9	Dep_Time_m	int64	The minute of departure		departure minute, not important since it's periodic and Dep_Time_hr is enough
10	Arv_Time_h	int64	The hour of arrival	☑	arrival hour can have an effect(ex.: arriving late could be cheaper)
11	Arv_Time_m	int64	The minute of arrival		not important for the same reasons as Dep_Time_m
13	Duration_mins	int64	flight duration in minutes	☑	duration
14	Gas_Price	float64	Gas prices	☑	from 2nd dataset, examine how gas prices correlate to air ticket prices
15	Src_Temperature	float64	Temperature at source city	☑	from 3rd dataset, testing if a relation exist with air ticket prices
16	Dest_Temperature	float64	Temperature at destination city	☑	//

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.

In [55]:

```
target = ['Price']
predictors = [
'Airline',
'Source',
'Destination',
'Total_Stops',
'Dep_Time_h',
'Arv_Time_h',
'Duration_mins',
'Gas_Price',
'Src_Temperature',
'Dest_Temperature' ]
Index = ['Date_of_Journey']
```

In [56]:

```
final_dataset = combined_df[Index + predictors + target ].copy(deep=True)
final_dataset.head()
```

Out[56]:

	Date_of_Journey	Airline	Source	Destination	Total_Stops	Dep_Time_h	Arv_Time_h	Duration_mins
0	2019-03-24	IndiGo	Banglore	New Delhi	non-stop	22	1	17
1	2019-05-01	Air India	Kolkata	Banglore	2 stops	5	13	44
2	2019-06-09	Jet Airways	Delhi	Cochin	2 stops	9	4	114
3	2019-05-12	IndiGo	Kolkata	Banglore	1 stop	18	23	32
4	2019-03-01	IndiGo	Banglore	New Delhi	1 stop	16	21	28

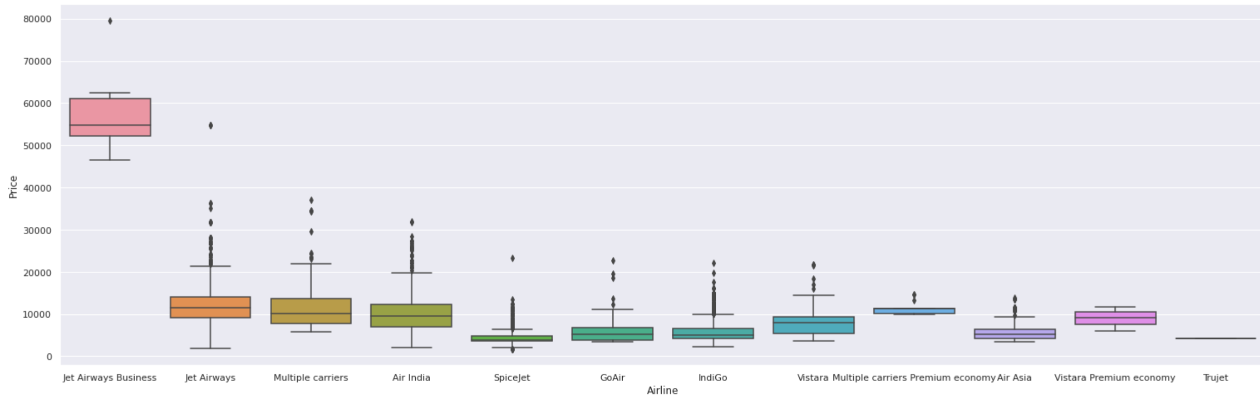
Airline & Flight Price

In [57]:

```
sns.set(rc = {'figure.figsize':(26,8)})
sns.boxplot(x='Airline', y='Price', data = flights_df.sort_values('Price', ascending = False))
```

Out[57]:

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



The plot shows that most of the airlines fall into a range of prices except for Jet Airways Business, which is significantly more expensive.

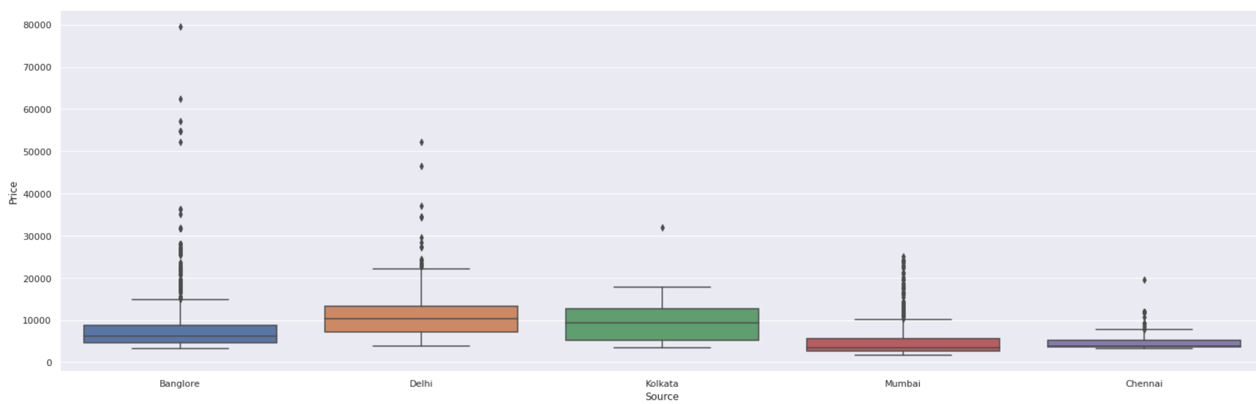
Source City and Price

In [58]:

```
sns.boxplot(x='Source', y='Price', data = final_dataset.sort_values('Price', ascending = False))
```

Out[58]:

```
<AxesSubplot:xlabel='Source', ylabel='Price'>
```



This plot doesn't show much variance between the cities, suggesting that maybe the source city is not a strong predictor of price, at least in this limited sample. Delhi has the heighest prices, and Banglore looks to have a very high outliers compared to the rest of source cities.

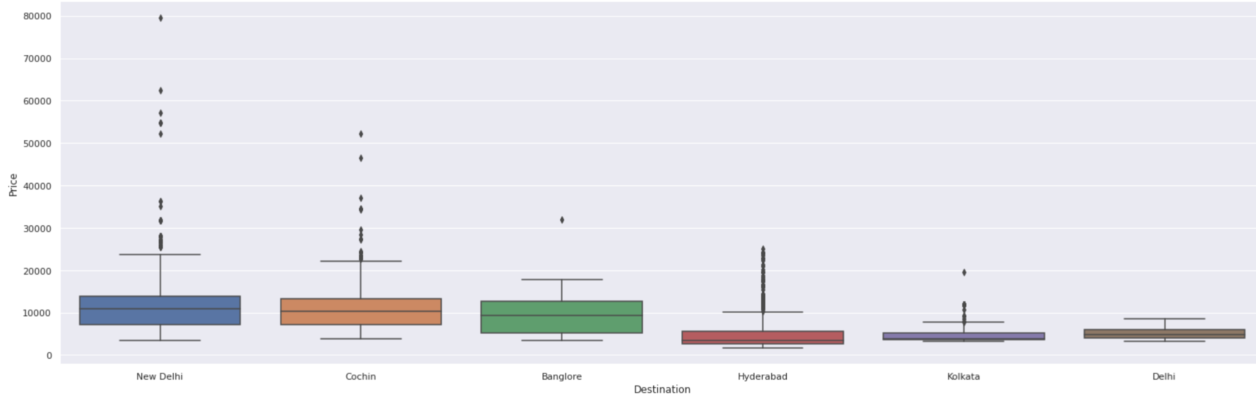
Destination City and Flight Price

In [59]:

```
sns.boxplot(x='Destination', y='Price', data = final_dataset.sort_values('Price', ascending = False))
```

Out[59]:

```
<AxesSubplot:xlabel='Destination', ylabel='Price'>
```

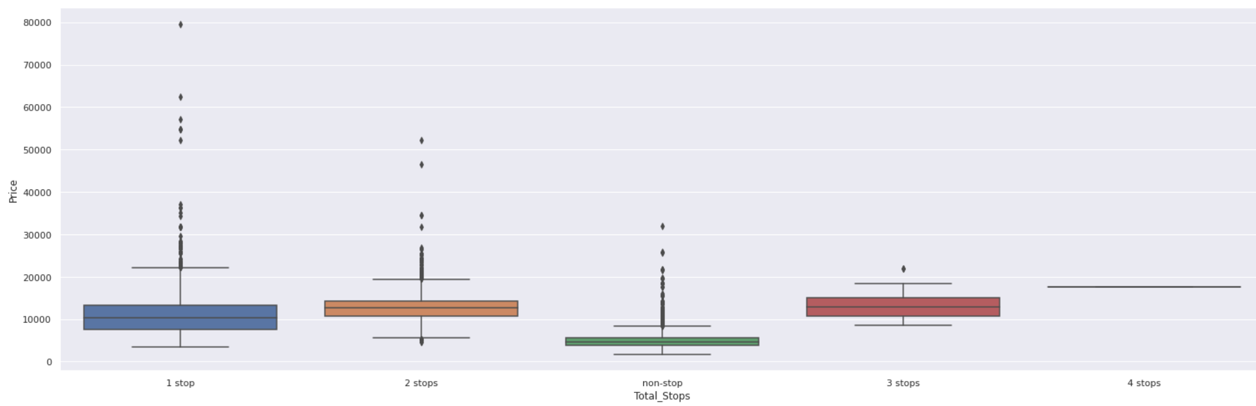


This plot doesn't show much variance between the destination cities as well, again I don't believe this fact could be generalized given that the sample of cities is so small. New Delhi has the heighest priced flights, with Cochin close second, and they have high outliers.

Total Number of Stops and Flight Price

In [60]:
```
sns.boxplot(x='Total_Stops', y='Price', data = final_dataset.sort_values('Price', ascending = False))
```

Out[60]:
```
<AxesSubplot:xlabel='Total_Stops', ylabel='Price'>
```

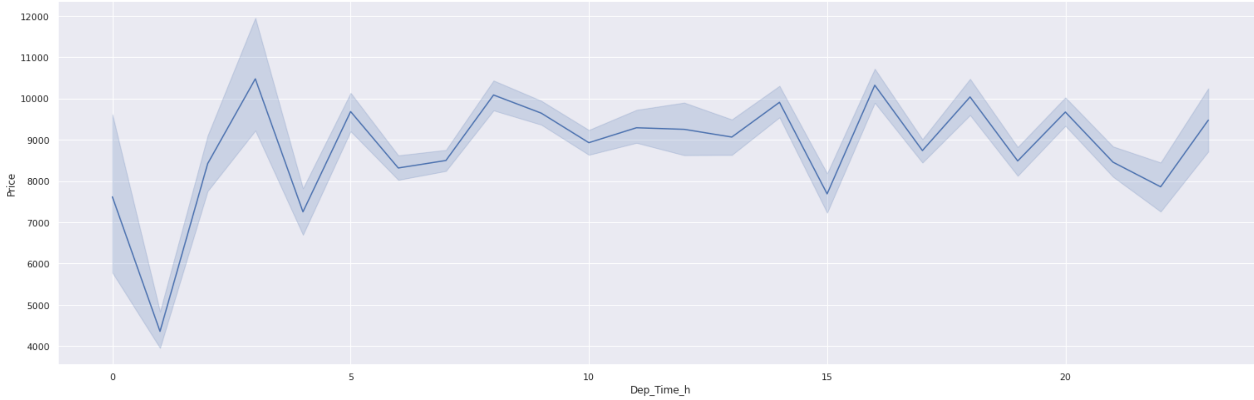


We can see a nice difference between the boxes in the plot. It is easy to see that the price seem to increase with the increase of stops, suggesting a direct correlation.

Hour of Departure and Flight Price

In [61]:
```
sns.lineplot(x='Dep_Time_h', y='Price', data = final_dataset)
```

Out[61]:
```
<AxesSubplot:xlabel='Dep_Time_h', ylabel='Price'>
```



This plot shows that the lowest priced flights depart close to midnight, with the lowest around 1am. The price climbes to the peak around 4am and then for the rest of the day it fluctuates in a relativily small range, with a sharp drop around the afternoon.

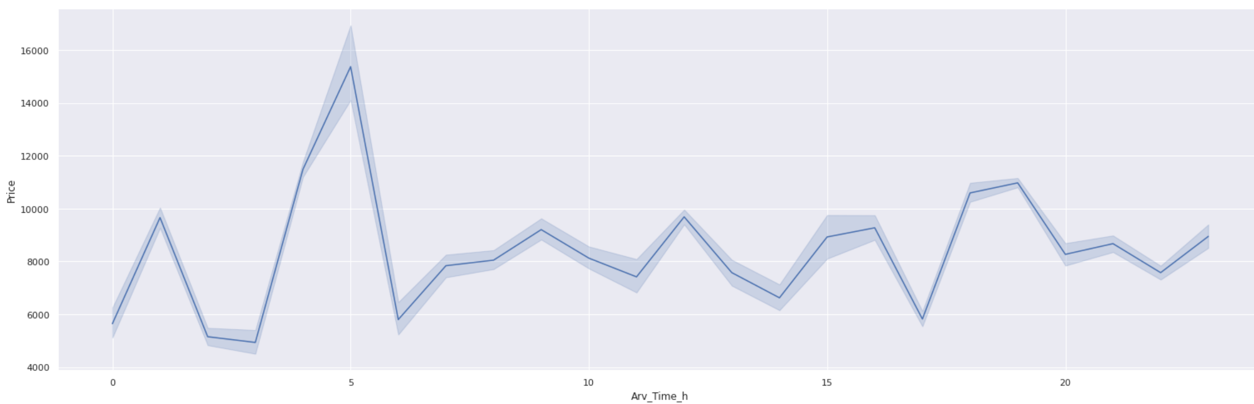
Time of Arrival and Flight Price

In [62]:

```
sns.lineplot(x='Arv_Time_h', y='Price', data = final_dataset)
```

Out[62]:

```
<AxesSubplot:xlabel='Arv_Time_h', ylabel='Price'>
```



This plot shows that the lowest priced flights arrive to destination between midnight and 3 am. The peak price in around sunrise and the start of the day (5 am). The price drops and stays in a relativily small range, with almost periodic local peaks and valleys.

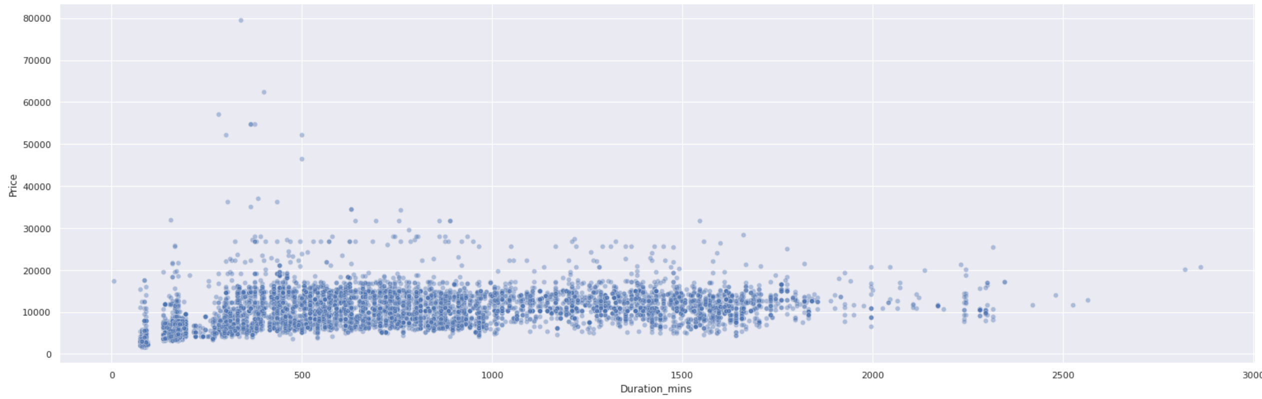
Duration and Flight Price

In [63]:

```
sns.scatterplot(x='Duration_mins', y='Price', data = final_dataset,alpha=0.4)
```

Out[63]:

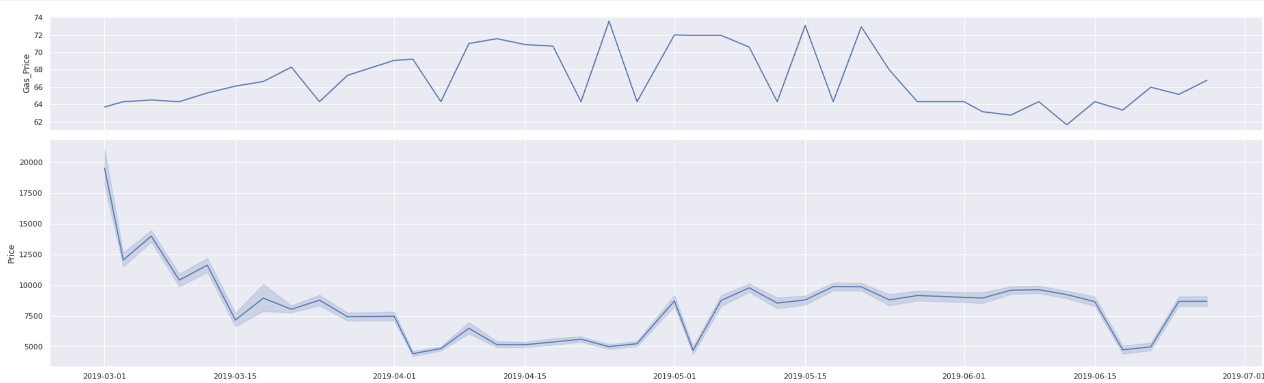
```
<AxesSubplot:xlabel='Duration_mins', ylabel='Price'>
```



The scatter plot above shows there isn't much variance in the price for based on duration. We can notice that the density of flights with shorter duration is higher. Flights with shorter duration also tend to have some high outliers.

Now we examine predictors from the additional datasets against the target variable from the original dataset

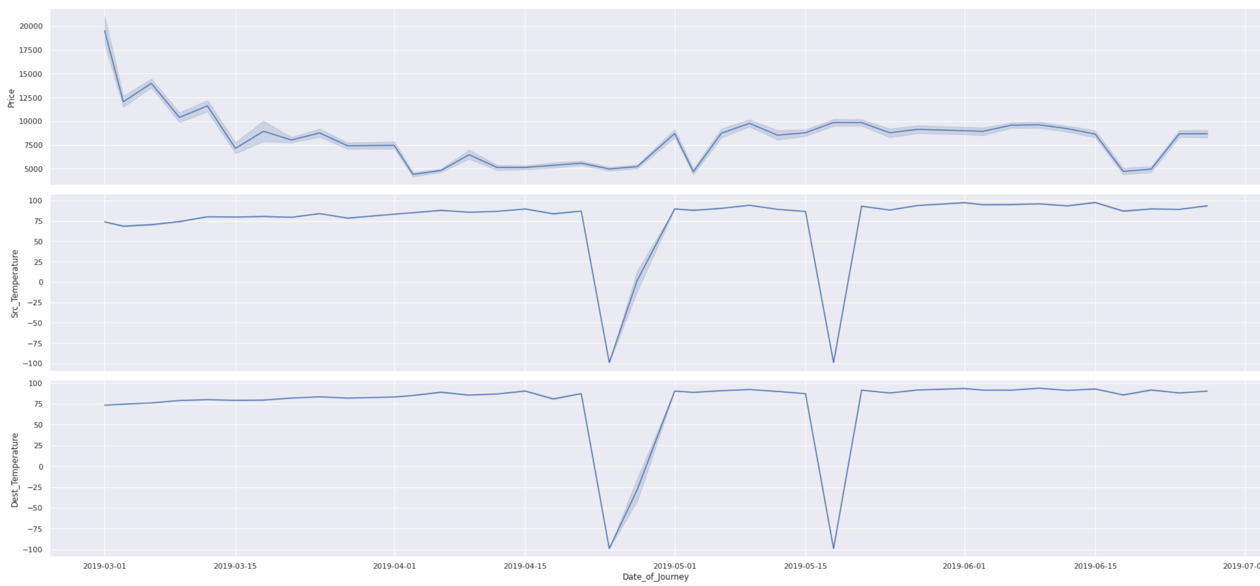
In [64]:
```
f, axs = plt.subplots(2,1,
                      sharex=True,
                      gridspec_kw=dict(height_ratios=[1,2]))
sns.lineplot(data= final_dataset,
                x="Date_of_Journey", y = 'Gas_Price',
                ax=axs[0],
                legend=False)
sns.lineplot(data= final_dataset,
                x="Date_of_Journey", y="Price",
                ax=axs[1] )
f.tight_layout()
```



This plot compares the price of flights and price of gas over time on the same x-axis. Surprisingly, the price of flights seem to move in the inverse direction of the price of gas. As gas prices drop, airfares climb up and same with the opposite. This suggests that they could be inversely proportional.

In [65]:
```
f, axs = plt.subplots(3,1,
                    figsize=((26,12)),
                    sharex=True,
```

```
                    gridspec_kw=dict(height_ratios=[1,1,1]))

sns.lineplot(data= final_dataset,
                x="Date_of_Journey", y = 'Price',
                ax=axs[0],
                legend=False)

sns.lineplot(data= final_dataset,
                x="Date_of_Journey", y="Src_Temperature",
                ax=axs[1] )

sns.lineplot(data= final_dataset,
                x="Date_of_Journey", y="Dest_Temperature",
                ax=axs[2] )

f.tight_layout()
```



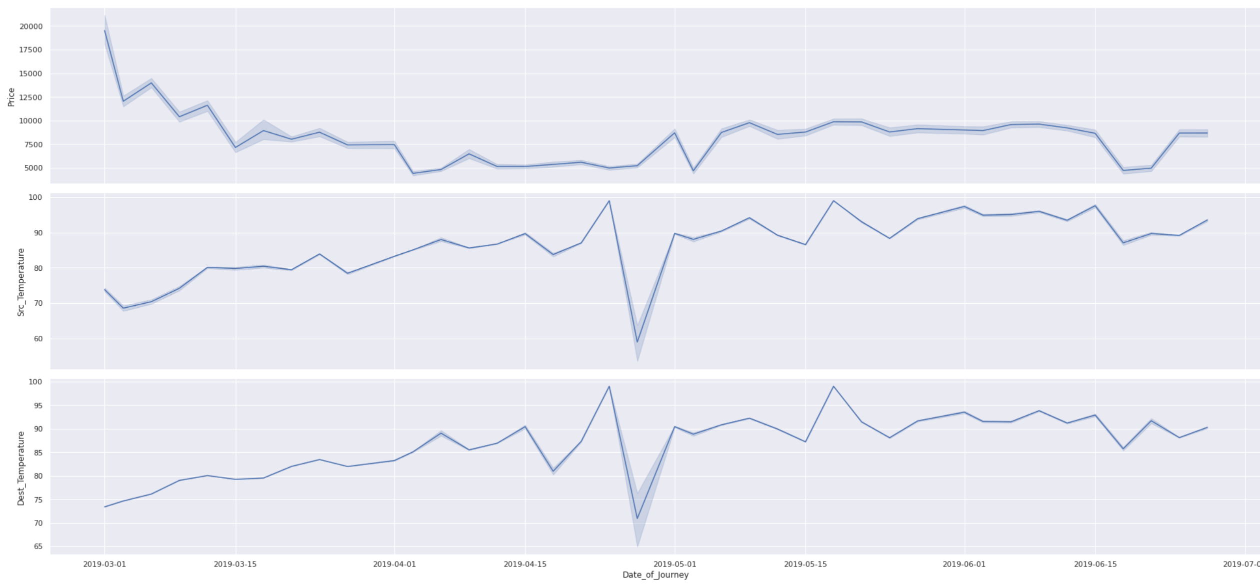
The plot above draws source and destination temperatues as well as prices over time on one axis. Unfortunately, there seem to be a problem with the temperature data, there are two points per each column that has an element set to -99F. The missed up data point make it harder to see patterns.

This is a speculation, but this could be 99F that was turned negative somehow. Anyway, 99F seems reasonable giving the neighboring values, also there are no negative values so I will go simple and just do abs().

In [66]:
```
final_dataset['Dest_Temperature'] = np.abs(final_dataset['Dest_Temperature'])
final_dataset['Src_Temperature'] = np.abs(final_dataset['Src_Temperature'])
```

In [67]:
```
f2, axs = plt.subplots(3,1,
                    figsize=((26,12)),
                    sharex=True,
                    gridspec_kw=dict(height_ratios=[1,1,1]))
```

```
sns.lineplot(data= final_dataset,
                x="Date_of_Journey", y = 'Price',
                ax=axs[0],
                legend=False)

sns.lineplot(data= final_dataset,
                x="Date_of_Journey", y="Src_Temperature",
                ax=axs[1] )

sns.lineplot(data= final_dataset,
                x="Date_of_Journey", y="Dest_Temperature",
                ax=axs[2] )

f2.tight_layout()
```



Much better! Ok, now we can see a slight negative correlation between temperature and flight prices. In other words, flights are cheaper on hot days that cooler ones. This is another interesting observation gained from the datasets merging.