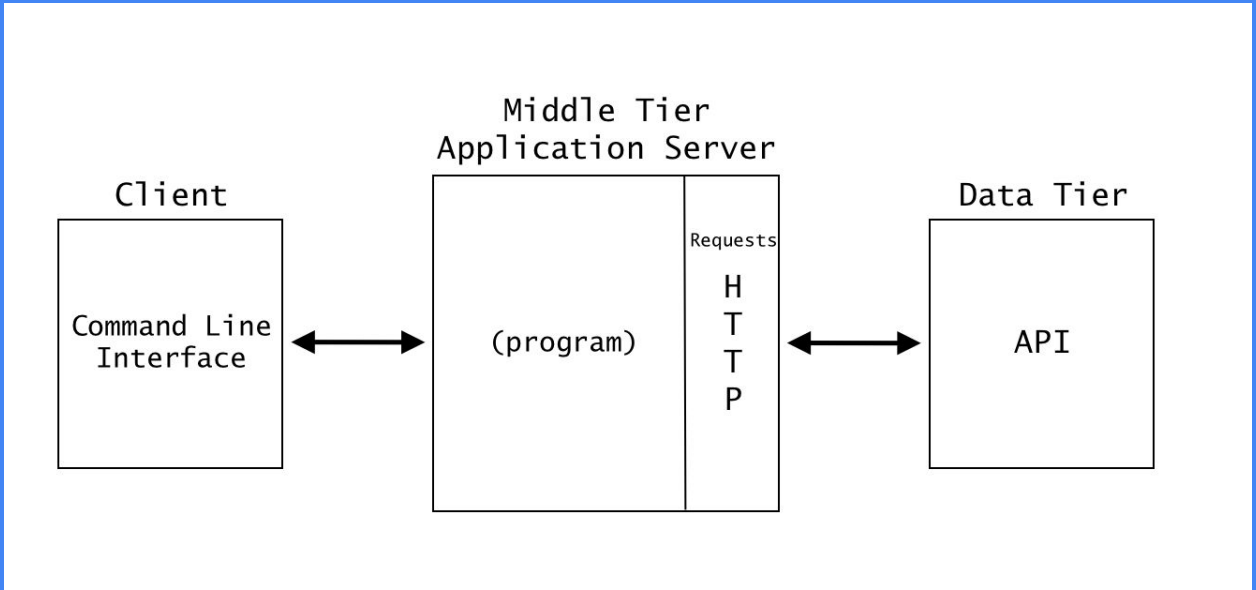
Flight-price-comparison project.

Flight price comparison

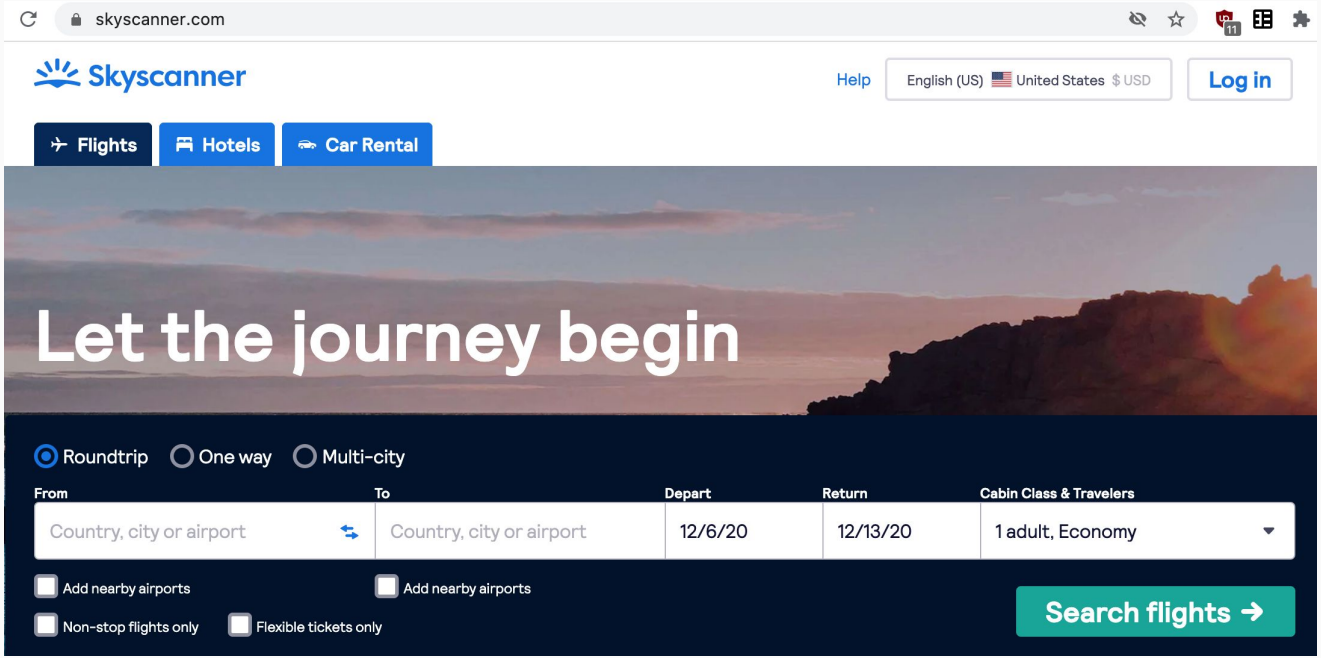
Parsa Falah-Adl, Nalini Suresh, Brenda Wang,

Architecture Diagram (parsa)



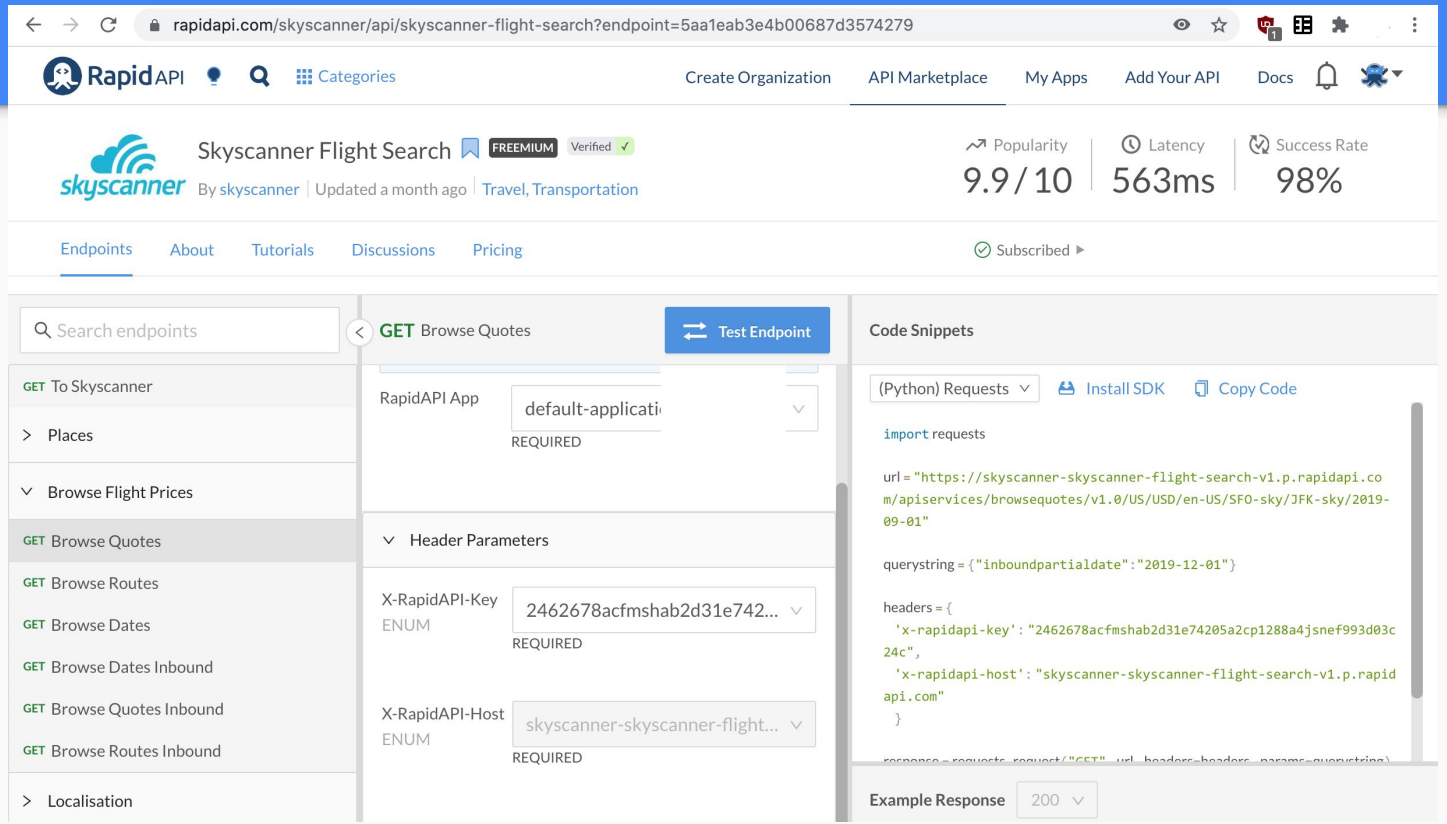
SkyScanner

"Skyscanner is a metasearch engine and travel agency based in Edinburgh, Scotland and owned by Trip.com Group, the largest online travel agency in China. The site is available in over 30 languages and is used by 100 million people per month." (parsa)



Rapid API

Provides their key to allow us free access to the Skyscanner API.



Summary

- Accessed Skyscanner's airfare dataset through Rapid API.
- Parsed the data and extracted data for each flight ticket (price, departure date, flight type - direct or indirect)
- Recorded the data in a nested dictionary
  - { dayOfTheWeek : {direct : $, indirect : $} }
- Present the data in a clustered bar graph to allow comparison of prices.
- Python libraries used: HTTP Requests, regular expressions, JSON, pandas, matplotlib, numpy

```
import urllib
import json
import urllib.request, urllib.parse, urllib.error
import ssl
```

```
import requests
import re
import pandas as pd

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

Requests

Flight search parameters are a part of the URL:

1. User market
2. Currency
3. ISO language
4. Departing Location
5. Arrival Location
6. Departure time period
7. Optional: Return Flight time period

Headers contain the API key

These are passed to request method to communicate with the web page

Returned data contained in response.text

```
#SFO -> London Airports for January 2021
url = "https://skyscanner-skyscanner-flight-search-v1.p.rapidapi.com/apiservices/browsequotes/v1.0/US/USD/en-US/SFO-sky/LOND-sky/2021-01"

headers = {
    'x-rapidapi-key': "2462678acfmshab2d31e74205a2cp1288a4jsnef993d03c24c",
    'x-rapidapi-host': "skyscanner-skyscanner-flight-search-v1.p.rapidapi.com"
    }

response = requests.request("GET", url, headers=headers)

SFO_LOND_jan =  response.text.split('}, {\n')
```

What the data looked like

print(response.text)

Contains the following:

Each quote is numbered.

For each quote ID:

Min price for the flight, Direct or Indirect, Carrier ID, Originating and destination ID, and departure date.

There was also more data at the end that defined values for Carrier ID and Destination ID, but we were not concerned with that.


```
{
  "Quotes" : [ {
    "QuoteId" : 1,
    "MinPrice" : 277,
    "Direct" : false,
    "OutboundLeg" : {
      "CarrierIds" : [ 1065 ],
      "OriginId" : 81727,
      "DestinationId" : 65698,
      "DepartureDate" : "2021-01-22T00:00:00"
    },
    "QuoteDateTime" : "2020-12-01T03:02:00"
  }, {
    "QuoteId" : 2,
    "MinPrice" : 278,
    "Direct" : false,
    "OutboundLeg" : {
      "CarrierIds" : [ 1065 ],
      "OriginId" : 81727,
      "DestinationId" : 82398,
      "DepartureDate" : "2021-01-18T00:00:00"
    },
    "QuoteDateTime" : "2020-11-30T21:35:00"
  }, {
    "QuoteId" : 3,
    "MinPrice" : 278,
    "Direct" : false,
    "OutboundLeg" : {
      "CarrierIds" : [ 1065 ],
      "OriginId" : 81727,
      "DestinationId" : 82398,
      "DepartureDate" : "2021-01-25T00:00:00"
    },
    "QuoteDateTime" : "2020-11-30T22:47:00"
  }, {
    "QuoteId" : 4,
    "MinPrice" : 283,
    "Direct" : true,
    "OutboundLeg" : {
      "CarrierIds" : [ 1218 ],
      "OriginId" : 81727,
      "DestinationId" : 65698,
      "DepartureDate" : "2021-01-18T00:00:00"
    },
    "QuoteDateTime" : "2020-12-01T10:44:00"
```

Data Structure: Dictionary

Set up the structure for the nested dictionaries:

1. Value = counts
2. Value = prices
    a. Adds the price and divides by the corresponding value in the count dict to get the average price.

```
SFO_LOND_jan_ave_price = {0 : {"direct" : 0, "indirect" : 0},
                          1 : {"direct" : 0, "indirect" : 0},
                          2 : {"direct" : 0, "indirect" : 0},
                          3 : {"direct" : 0, "indirect" : 0},
                          4 : {"direct" : 0, "indirect" : 0},
                          5 : {"direct" : 0, "indirect" : 0},
                          6 : {"direct" : 0, "indirect" : 0}}
```

2 Ways to Parse and Extract Data

- JSON
- Regular Expressions

JSON

Using json.loads(), a Python list is created of the individual quote data.

Each item within the list is a dictionary with keys/value pair.

```
dict_keys(['Quotes', 'Carriers', 'Places',
'Currencies'])

Key "Quotes" contains the following information:

{'QuoteId': 1, 'MinPrice': 551, 'Direct': False,
'OutboundLeg': {'CarrierIds': [1324], 'OriginId':
81727, 'DestinationId': 48018, 'DepartureDate':
'2021-01-18T00:00:00'}, 'QuoteDateTime':
'2020-12-15T21:29:00'}
```

```
# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE
```

```
#  Load the JSON
try:
    js = json.loads(data.text)
except Exception as err:
    print(f"error: {err}")
    sys.exit(-1)

# Here is how you can look at your keys
print(js.keys())

#Here is how you retrieve data from a spacific key
print(js['Quotes'])
```

```
27, 'DestinationId': 48018, 'DepartureDate': '2021-01-13T00:00:00'}, 'QuoteDateTi
```

JSON (cont.)

```
quotes = js['Quotes']
```

The quotes list is traversable with a **for** loop, and the values corresponding to the ‘MinPrice’, ‘Direct’, and ‘QuoteDateTime’ keys are extracted.

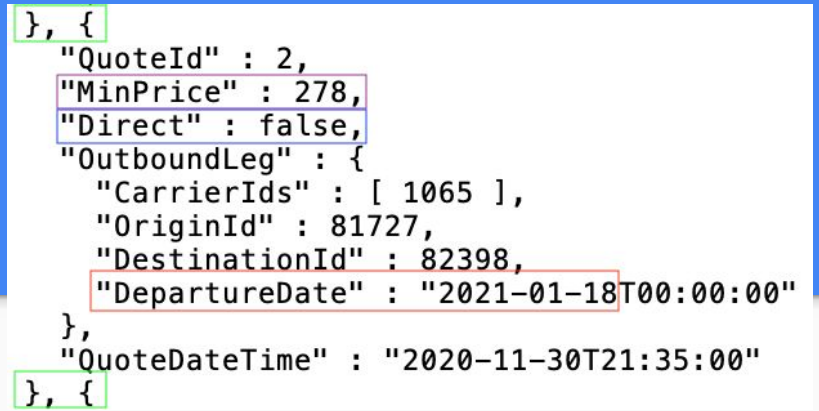
Updates the total count variable for the day.

Calculate the total price for the day.

Calculate the average price for the day.

```
total_flights = 0

for q in quotes:
  print(q['MinPrice'])
  total_flights +=1

  # update total flight count for that day

  dt = q['QuoteDateTime']
  d = dt.split(':')
  day = (int(d[-2]))%7

  flight_info[day][2] +=1

  # Add count to correct day for dir or undir

  if q['Direct'] == 'true':
    flight_info[day][0] =  flight_info[day][0] + (q['MinPrice']-flight_info[day][
  else:
    flight_info[day][1] =  flight_info[day][1] + (q['MinPrice']-flight_info[day][
```

Regular Expressions



The string data in response.text was split by the curly brackets that separated individual quotes into a list of strings, and a **for** loop was used to parse the the quotes in the list.

Used re's findall method to search for the data header + data point in each quote, with parentheses to extract the desired data point into an individual list.

```
SFO_LOND_jan =  response.text.split('}, {\n')
```

```
for quote in SFO_LOND_jan:
    x = re.findall('"MinPrice" : ([0-9]+),', quote)
    y = re.findall('"Direct" : ([ft].+),', quote)
    z = re.findall('"DepartureDate" : "([0-9]{4}-[0-9]{2}-[0-9]{2})', quote)
```

Adding key, (key, value) pairs to the nested dictionary

```
for a in x:
    price = a

for b in y:
    if b == "true":
        flight = "direct"
    elif b == "false":
        flight = "indirect"

for c in z:
    day = pd.Timestamp(c)
    dpt_day = day.dayofweek
```

(This is in the same **for** block that parses individual quotes list.)

Variables: price, flight, and dpt_day are used to update the count dictionary and the price dictionary.

First Key: the departure day (dpt_day)

Second Key: “direct” or “indirect”

```
SFO_LOND_jan_count[dpt_day][flight] += 1

SFO_LOND_jan_ave_price[dpt_day][flight] += (int(price) / SFO_LOND_jan_count[dpt_day][flight])
```

Pandas

Used to easily convert the extracted departure dates to a value for the day of the week (0: Monday - 6: Sunday)

Object of Timestamp class is created by passing a date in the format of YYYY-MM-DD and the dayofweek method converts it to a digit.

```
z = re.findall('"DepartureDate" : "([0-9]{4}-[0-9]{2}-[0-9]{2})', quote)

for c in z:
    day = pd.Timestamp(c)
    dpt_day = day.dayofweek
```

Data Structure: Function

The 0-6 digit value for day of the week is passed to this function to get the English name when printing the chart.

```
def dayOfWeek(d):
    if d == 0:
        day = "Monday"
    elif d == 1:
        day = "Tuesday"
    elif d == 2:
        day = "Wednesday"
    elif d == 3:
        day = "Thursday"
    elif d == 4:
        day = "Friday"
    elif d == 5:
        day = "Saturday"
    else:
        day = "Sunday"
    return day
```

Data Structure: List

The price dictionary's dayOfTheWeek keys are parsed and their values (a nested key, value pair) is saved as a flights dictionary object.

Then, each flights' key is parsed ("direct" or "indirect"), and their value (average price) is appended to separate lists.

The lists are used to print a chart of the average prices and to pass the prices in the correct order when creating the bar graph.

```
SFO_LOND_direct = list()
SFO_LOND_indirect = list()

for day in SFO_LOND_jan_ave_price:
    print("Day of week: ", dayOfWeek(day))

    flights = SFO_LOND_jan_ave_price[day]

    direct = round(flights["direct"], 2)
    SFO_LOND_direct.append(direct)

    indirect = round(flights["indirect"], 2)
    SFO_LOND_indirect.append(indirect)

    print("Direct flight average price: $", direct)
    print("Indirect flight average price: $", indirect)
    print()
```

matplotlib, numpy

Numpy's arange method sets the positions of the bar on the x-axis.

For each day of the week on the x-axis, there is a data point for the direct flight and another for the indirect flight.

The y-axis shows the price in USD.

Matplotlib creates and customizes the diagram.

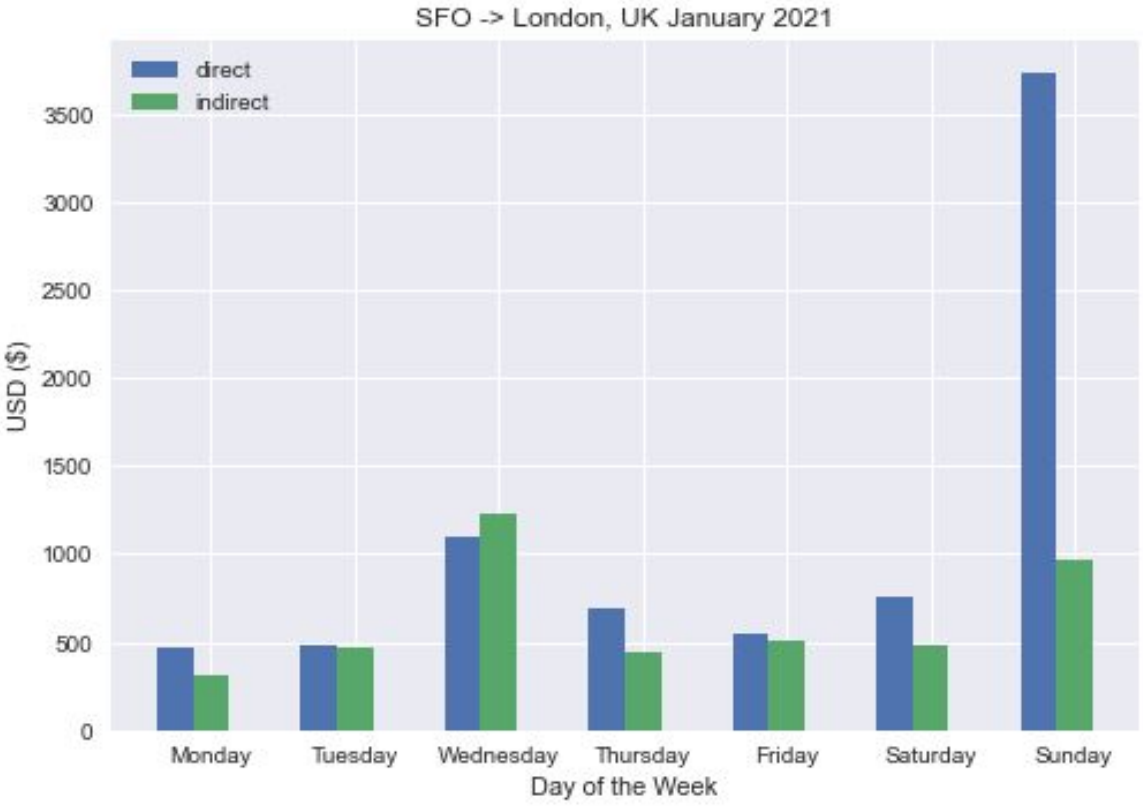
```
#set bar width and heights
barWidth = 0.25
bars1 = SFO_LOND_direct
bars2 = SFO_LOND_indirect

#set position of bar on x-axis
r1 = np.arange(len(bars1))
r2 = [x + barWidth for x in r1]

#make the plot
plt.style.use('seaborn')
plt.bar(r1, bars1, width=barWidth, label='direct')
plt.bar(r2, bars2, width=barWidth, label='indirect')

#add axis ticks and labels
plt.xticks([r + barWidth for r in range(len(bars1))],
["Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday", "Sunday"])
plt.xlabel('Day of the Week')
plt.ylabel('USD ($)')
plt.title('SFO -> London, UK January 2021')

#create legend and show graphic
plt.legend()
plt.show()
```

Routes Analyzed:

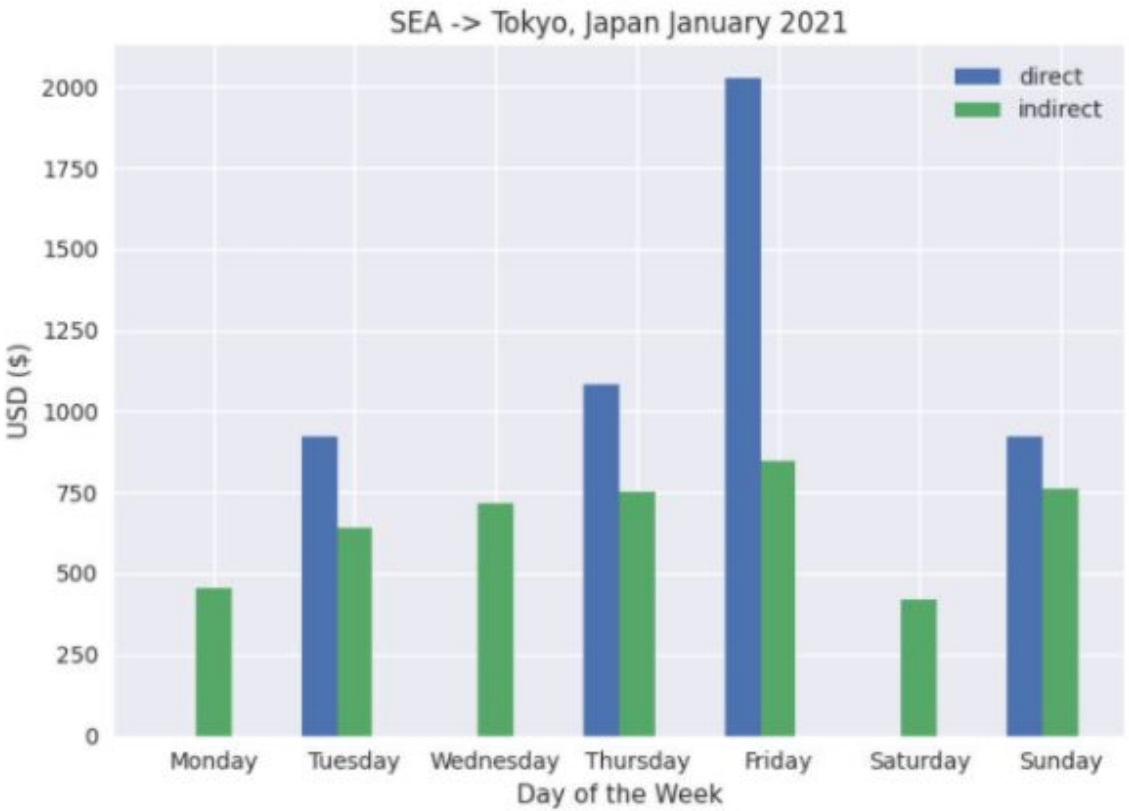
for time period
2021 JANUARY 1-31

- SFO -> London (2 Int'l Airports)
- SEA -> Tokyo
- SFO -> Dubai

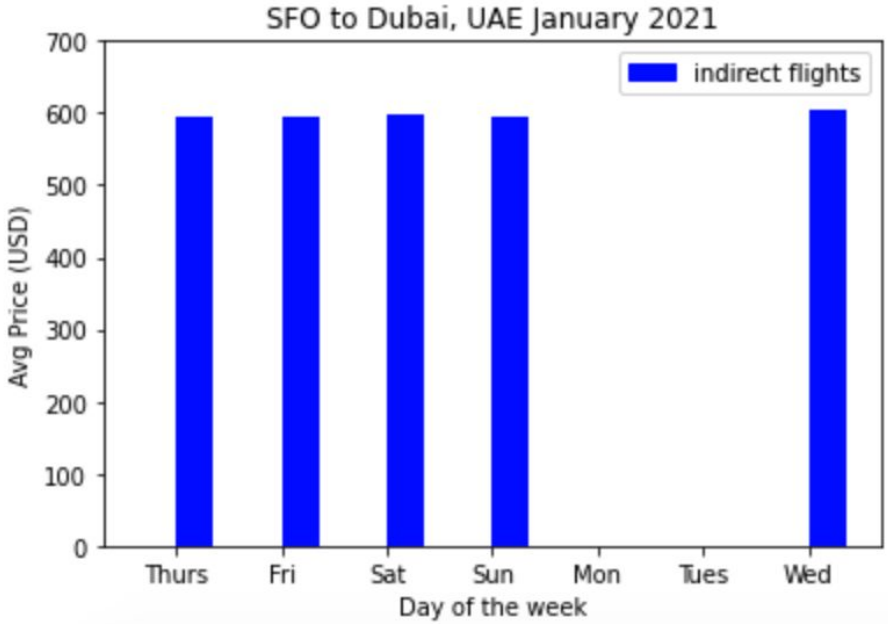
*note: the available flights and prices may have changed after the quotation date (when we first accessed them)




```
Day of week:  Monday
Direct flight average price: $ 473.5
Indirect flight average price: $ 305.0

Day of week:  Tuesday
Direct flight average price: $ 480.0
Indirect flight average price: $ 469.5

Day of week:  Wednesday
Direct flight average price: $ 1103.17
Indirect flight average price: $ 1227.0

Day of week:  Thursday
Direct flight average price: $ 690.0
Indirect flight average price: $ 443.5

Day of week:  Friday
Direct flight average price: $ 547.0
Indirect flight average price: $ 508.33

Day of week:  Saturday
Direct flight average price: $ 762.25
Indirect flight average price: $ 477.0

Day of week:  Sunday
Direct flight average price: $ 3731.48
Indirect flight average price: $ 962.22
```




```
Day of week:  Monday
Direct flight average price: $ N/A
Indirect flight average price: $ 454.0

Day of week:  Tuesday
Direct flight average price: $ 923.5
Indirect flight average price: $ 639.5

Day of week:  Wednesday
Direct flight average price: $ N/A
Indirect flight average price: $ 714.0

Day of week:  Thursday
Direct flight average price: $ 1081.5
Indirect flight average price: $ 750.5

Day of week:  Friday
Direct flight average price: $ 2025.55
Indirect flight average price: $ 846.5

Day of week:  Saturday
Direct flight average price: $ N/A
Indirect flight average price: $ 421.0

Day of week:  Sunday
Direct flight average price: $ 924.0
Indirect flight average price: $ 760.5
```




```
Day of the Week:  Thursday
Indirect flight average price: $ 596.0

Day of the Week:  Friday
Indirect flight average price: $ 594.5

Day of the Week:  Saturday
Indirect flight average price: $ 599.5

Day of the Week:  Sunday
Indirect flight average price: $ 594.0

Day of the Week:  Monday
Indirect flight average price: $ 0

Day of the Week:  Tuesday
Indirect flight average price: $ 0

Day of the Week:  Wednesday
Indirect flight average price: $ 604.5
```

Comparison of JSON and regex

Both work acceptably in this situation because the data wasn't too complex; we only needed data from 3 keys.

```
for q in quotes:
  print(q['MinPrice'])
  total_flights +=1

  # update total flight count for that day

  dt = q['QuoteDateTime']
  d = dt.split(':')
  day = (int(d[-2]))%7

  flight_info[day][2] +=1

  # Add count to correct day for dir or undir

  if q['Direct'] == 'true':
    flight_info[day][0] =  flight_info[day][0] + (q['MinPrice']-flight_info[day][0])/i
  else:
    flight_info[day][1] =  flight_info[day][1] + (q['MinPrice']-flight_info[day][1])/i
```

```
for quote in SFO_LOND_jan:
    x = re.findall('"MinPrice" : ([0-9]+),', quote)
    y = re.findall('"Direct" : ([ft].+),', quote)
    z = re.findall('"DepartureDate" : "([0-9]{4}-[0-9]{2}-[0-9]{2})', quote)
```

Improvements

- Increase the quote date range to obtain a larger data set
- Choose only route that includes both direct and indirect flights
- Allow user to choose the departure and arrival destinations and analyze data for them

Conclusions from our Data and Applications

If, in January 2021, you wanted to take a one-way trip to a city in another country, of these three options, the cheapest flight on average would be indirect to Tokyo, departing on Saturday, for $421.00.

This code can be used to find and sort average airfare prices for any upcoming route, as provided by SkyScanner's API, and present the data in a graph that facilitates comparison of direct or not by departure date.