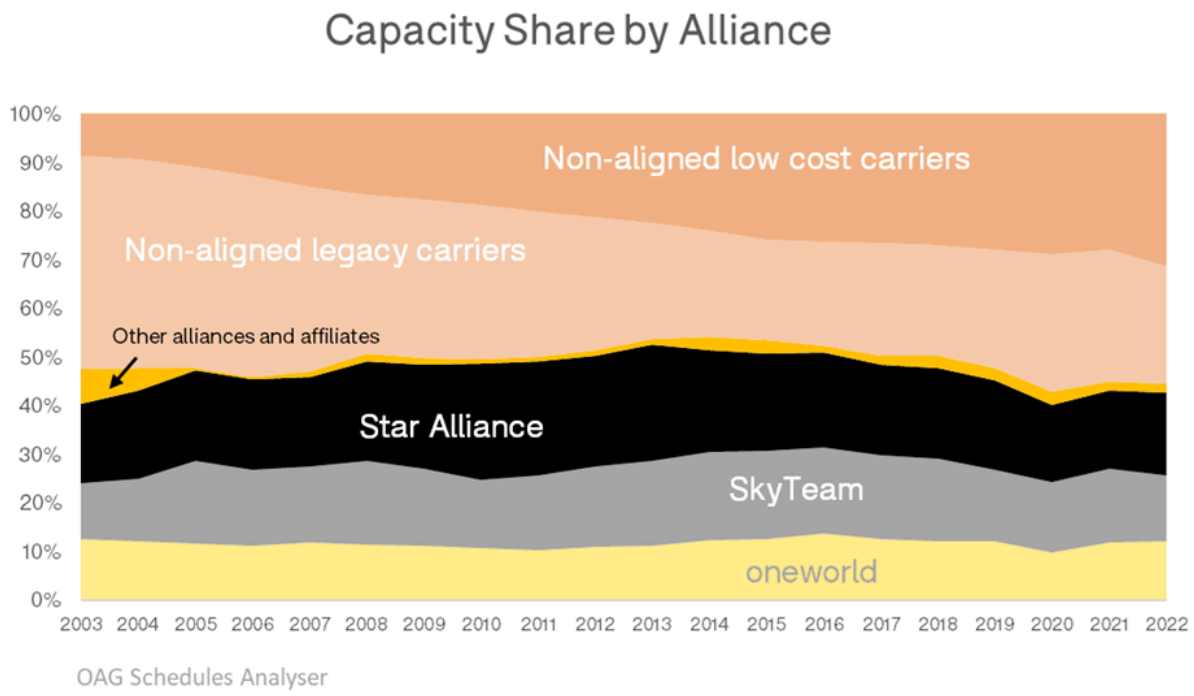
Open in Colab

Airline Text Analytics

Airline companies are in fierce competition. One key dimension of competition is customer service, a direct driver of customer satisfaction.



As part of an effort to better understand passenger satisfaction within the Star Alliance network, we analyze customer reviews from thousands of airline passengers. The goal is to identify how factors like sentiment, seat type, travel purpose, and specific routes relate to overall recommendation rates.

The dataset includes 23,000+ reviews, with key variables such as:

- `Airline Name` : the name of the airline.
- `Route` : the flight route taken by the passenger.
- `Review` : free-text review written by the traveler.
- `Type Of Traveller` : e.g., Solo Leisure, Business, Couple.
- `Recommended` : whether the traveler recommends the airline (yes/no).
- `Seat Type` : seat class (e.g., Economy, Business).

Data Source: Airlinequality.com via Kaggle

Required packages — run once before starting:

```
pip install pandas matplotlib seaborn nltk tqdm wordcloud transformers torch google-genai
# or with conda:
# conda install pandas matplotlib seaborn nltk tqdm && pip install wordcloud transformers torch
google-genai
```

Part I: Working with Text Data

Strings

Text data always starts out in the form of a string. A **string** is a sequence (list) of **characters**. This includes the (English) alphanumeric characters and also special characters like white space, punctuation, etc. Other symbols, like emoticons, can also appear in your data, specially in social networks data. Besides that, you can also find letters from other languages (Spanish, Portuguese, etc) or alphabets (Cyrillic, hiragana, etc), and even ideographs (such as Han characters).

Characters as numbers

There is a basic set of 127 characters, called the **ASCII characters**, which are encoded in the same way by all the computers, so you will never have trouble with them. They include the English letters (without accents), the numbers, basic punctuation (not curly quote marks or long dashes), white space, **control characters** such as the new line, represented in programming languages as `\n`, and other symbols familiar to you, such as the dollar ($) and the hash (#) symbols. The complete list can be easily found in Internet.

Non-ASCII characters can be encoded by different computers or different text editors in different ways. Mind that, if you capture string data on your own, you will probably find some of these characters in your data. Even when the documents are expected to be in English, they can be contaminated by other languages: Han characters, German dieresis, Spanish eñe, etc.

The preferred **encoding** is **UTF-8 (utf-8)**, which is the default encoding in Macintosh computers. Reading and writing text files in Pandas, the argument encoding allows you to manage both UTF-8 and the alternative encoding **Latin-1 (latin1)**. Windows computers use their own system, which is region specific. In US and Western Europe, this is **Windows-1252**, which is very close to Latin-1, though not exactly the same.

In [64]:
```
the_letter_A = chr(65)  # this
the_letter_A = 'A'      # is the same as this

print(the_letter_A)

# This one is in Farsi, it may not render on your computer
print(chr(0x0604))

# This one is tradtional chinese
print(chr(0x9F8D))
```

```
A
؄
龍
```

Strings as lists

Both lists and strings are **sequences**, so they share some core behaviours. A string can be thought of as a list of characters. You can use `len()` and slicing just like a list:

In [65]:
```
school = 'IESE Business School'
print(len(school))
print(school[:4])    # first 4 characters
print(school[-6:])   # last 6 characters
```

```
20
IESE
School
```

Python string methods

Python has a rich set of methods for manipulating strings:

Method	What it does
`lower()` / `upper()`	Change case
`replace(old, new)`	Find and replace
`split(sep)`	Split into a list of parts
`join(list)`	Join a list back into a string
`count(pattern)`	Count occurrences

In [66]:
```
name = 'Enric Junque de Fortuny'

print(name.upper())
print(name.lower())
print(name.split(' '))
print('---'.join(name.split(' ')))
```

```
ENRIC JUNQUE DE FORTUNY
enric junque de fortuny
['Enric', 'Junque', 'de', 'Fortuny']
Enric---Junque---de---Fortuny
```

Airline Reviews Dataset

Let's load the data. I've already partially cleaned it up for you (no duplicates). You can download it from here.

In [67]:
```
import pandas as pd

pd.set_option('display.max_colwidth', None)

## Use your local copy:
df = pd.read_csv('../resources/text/Airline_Reviews_v2.csv')[['Airline Name','From','To','Via','Review','Type
## Or download every time:
# df = pd.read_csv('https://raw.githubusercontent.com/ciri/iese-dsfb/refs/heads/main/resources/text/Airline_R

df.head(2)
```

Out[67]:

	Airline Name	From	To	Via	Review	Type Of Traveller	Recommended	Seat Type
0	AB Aviation	Moroni	Moheli	NaN	Moroni to Moheli. Turned out to be a pretty decent airline. Online booking worked well, checkin and boarding was fine and the plane looked well maintained. Its a very short flight - just 20 minutes or so so i didn't expect much but they still managed to hand our a bottle of water and some biscuits which i though was very nice. Both flights on time.	Solo Leisure	1	Economy Class
1	AB Aviation	Moroni	Anjouan	NaN	Moroni to Anjouan. It is a very small airline. My ticket advised me to turn up at 0800hrs which I did. There was confusion at this small airport. I was then directed to the office of AB Aviation which was still closed. It opened at 0900hrs and I was told that the flight had been put back to 1300hrs and that they had tried to contact me. This could not be true as they did not have my phone number. I was with a local guide and he had not been informed either. I presume that I was bumped off. The later flight did operate but as usual, there was confusion at check-in. The flight was only 30mins and there were no further problems. Not a good airline but it is the only one for Comoros.	Solo Leisure	0	Economy Class

Exploring the dataset

There is not much to `describe()` here since most columns are text. Let's use `.value_counts()` to count categories, and `.groupby()` to aggregate.

In [68]:
```
df['Seat Type'].value_counts()
```

Out[68]:
```
Seat Type
Economy Class       19129
Business Class       2097
Premium Economy       646
First Class           186
Name: count, dtype: int64
```

In [69]:
```
df.groupby('Seat Type').count()
```

Out[69]:

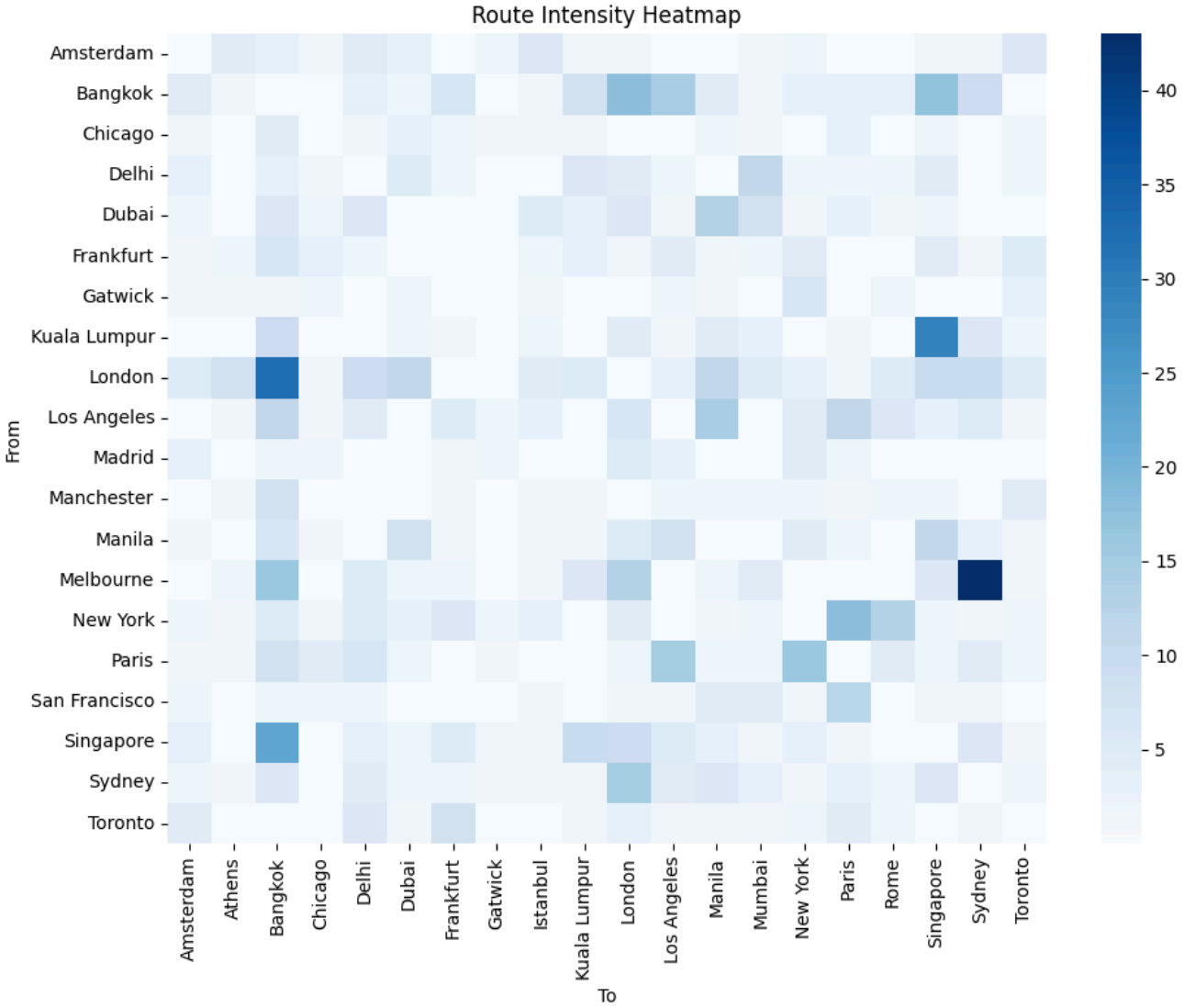
Seat Type	Airline Name	From	To	Via	Review	Type Of Traveller	Recommended
Business Class	2097	1806	1805	517	2097	1808	2097
Economy Class	19129	16747	16731	3533	19129	16829	19129
First Class	186	168	168	23	186	168	186
Premium Economy	646	608	605	145	646	613	646

In [71]:
```
df.groupby('Seat Type').nunique()
```

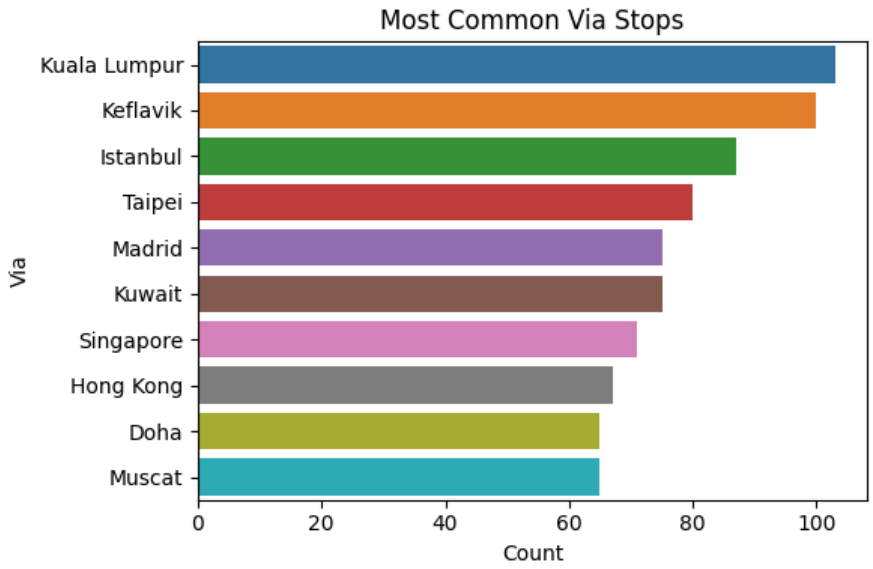
Out[71]:

Seat Type	Airline Name	From	To	Via	Review	Type Of Traveller	Recommended
Business Class	217	436	471	175	2097	4	2
Economy Class	489	2025	2271	677	19129	4	2
First Class	75	103	106	19	186	4	2
Premium Economy	157	240	260	83	646	4	2

In [72]:
```
import matplotlib.pyplot as plt
import seaborn as sns

routes = (
    df.dropna(subset=['From','To'])
      .groupby(['From','To'])
      .size()
      .reset_index(name='count')
)
top_from = routes['From'].value_counts().head(20).index
top_to   = routes['To'].value_counts().head(20).index

sub = routes[
    routes['From'].isin(top_from) &
    routes['To'].isin(top_to)
]
matrix = sub.pivot(index='From', columns='To', values='count').fillna(0)

plt.figure(figsize=(10,8))
sns.heatmap(matrix, cmap='Blues', vmin=0.0001)
plt.title('Route Intensity Heatmap')
plt.xlabel('To')
plt.ylabel('From')
plt.tight_layout()
plt.show()
```



In [73]:
```
top_via = df['Via'].value_counts().head(10)

plt.figure(figsize=(6,4))
sns.barplot(x=top_via.values, y=top_via.index)
plt.title('Most Common Via Stops')
plt.xlabel('Count')
plt.ylabel('Via')
plt.tight_layout()
plt.show()
```



Exploring the review text

When a Pandas column contains string, we can access its **string methods**. Pandas string methods are vectorized versions of the above methods. They return a series of the same length, in which each term results from applying a string function to the corresponding term of `s`. The syntax of these functions is typically `s.str.fname(args)`, where `s` is a Pandas series of strings (or a list of strings) and args is a set of ad hoc arguments.

For example, `.str.contains("pattern")` returns a Boolean series — True wherever the pattern appears:

In [74]:
```
f_has_space = df.Review.str.lower().str.contains("space")
df[f_has_space].head(3)
```

Out[74]:

	Airline Name	From	To	Via	Review	Type Of Traveller	Recommended	Seat Type
41	Adria Airways	Manchester	Ljubljana	NaN	Excellent flights both ways with Adria Airways from Manchester to Ljubljana. Staff friendly but unobtrusive. Leg room far better than most economy flights, seats comfy and plenty of storage space. Flights on time, quick and efficient boarding and disembarking, smooth take offs and landings. All in all a very pleasant experience.	Family Leisure	1	Economy Class
100	Aegean Airlines	Athens	Chania	NaN	We took 3 flights with Aegean and would not recommend them to anyone. Had 3 issues: 1. They overbooked our flight and bumped many people. We bought our flight 3 months ahead, but checked in about 18 hours before our flight (they allow check-in up to 24 hours ahead). We were issued a “standby” ticket without any seats. When we got to the gate, we were told we may or may not get onto the flight and if we didn’t then we would have to wait 4 hours for the late night flight, or try again in the morning. Lots of other people in the same situation. We were never issued boarding passes but we kept following up with the person at the ticket counter until they gave us passes, it seems like due to our persistence and nothing systematic. Luckily we got on that flight, but others didn’t. Nowhere in the policy does it say that whether or not you get on the flight is related to your check-in time. If you take Aegean, check in right away and be persistent about asking for seats! 2. They apply a checked bag policy at will. We were sitting near the gate. The gate attendants approached us and asked to weigh our bags. Our bags, which fit easily into any overhead bin (including previous Aegean flights) were found to be slightly overweight (apparently 9kgs are allowed, ours were about 10kgs). We asked if we could move 1kg into our purse / backpack. They said no, they need to check the two bags, and by the way they are charging us 40Euro per bag. This was for a 45 min flight from an island to Athens. We had no choice and were forced to pay. Also, there was no consistency in the way this rule was applied. We saw plenty of other people with carry-ons larger than ours allowed on the plane and there was plenty of space in the overhead bins. 3. Every flight we were on was late. Two of three were late by over 45 mins. We didn’t have any connections, but I imagine many connecting flights were missed. And, this didn’t happen to us, but it’s worth noting that when we landed in Athens, there were about 30 people waiting at the lost baggage desk. Most of them looked very angry. I asked one person what happened, and apparently an entire flight did not have their bags transferred at a connection. So not only will they force you to check your carry-on and then pay for a checked bag, they might lose it too. All-in-all, this is a pretty bad track record for 3 short island hop flights.	Couple Leisure	0	Economy Class
111	Aegean Airlines	Mytilini	Athens	NaN	Check-in uneventful. Boarding a zoo with a rude employee denying me boarding, claiming some zone story. Upon boarding I found a number of passengers that were probably the right 'zone' since they were seated in the rear already. An old ATR POS that Aegean must likely keep to connect some small islands that cannot accommodate jets. Why they would use these consistently for the third largest Greek island is unclear to me. Once seated in 15C and having put my backpack above, I see the flight attendant pushing it to make space for someone's big suitcase. When I objected, she pulled it down	Business	0	Economy Class

Airline Name	From	To	Via	Review	Type Of Traveller	Recommended	Seat Type
				and told me in the rudest way to put it under the seat in front of me. Said attendant did not wear a name tag.			

In [75]:
```
# Let's read one full review
print(df.loc[41].Review)
```

```
Excellent flights both ways with Adria Airways from Manchester to Ljubljana. Staff friendly but unobtrusive.
Leg room far better than most economy flights, seats comfy and plenty of storage space. Flights on time, quic
k and efficient boarding and disembarking, smooth take offs and landings. All in all a very pleasant experien
ce.
```

You try it

1. Use `str.contains` to find the number of reviews which mention the following keywords: `'Crew'`, `'Bag'`, `'Passenger'`.
2. Calculate the total count across all 3 keywords.
3. Does review 186 from Aegean Airlines contain the word crew?

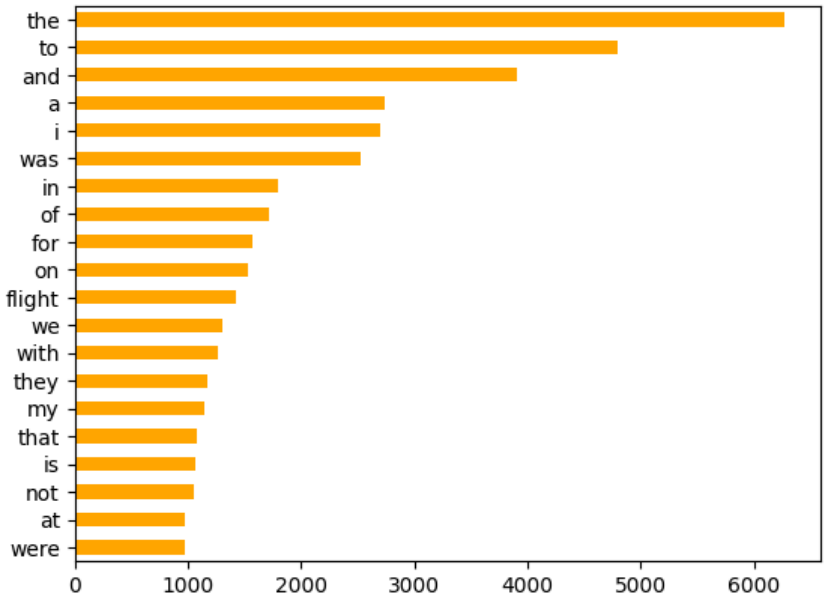
Hint: don't forget `.lower()`!

In [ ]:
```

```

Topic Detection: Frequent Word Analysis

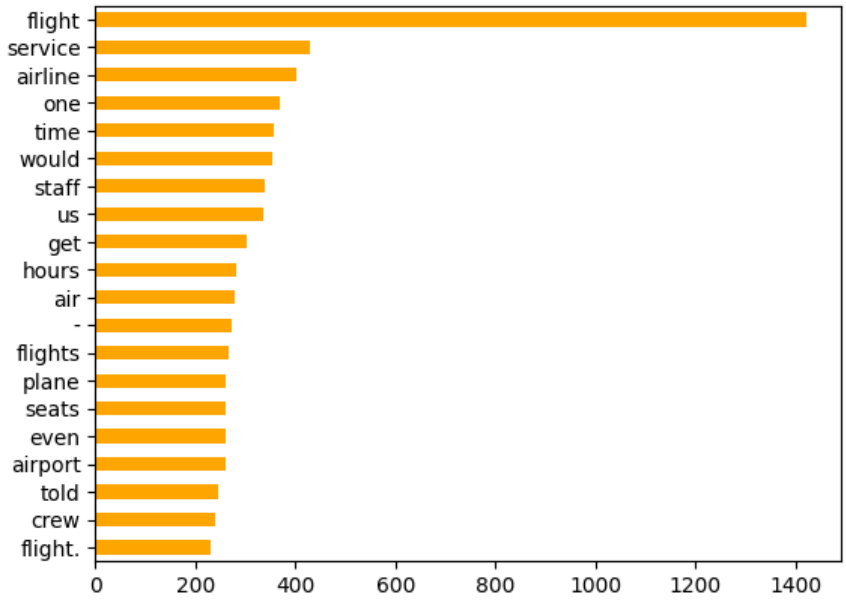
Advanced techniques like topic modeling (LDA) or transformer-based classification exist, but a simple and quick method is **word frequency analysis**. It gives a fast sense of recurring topics — service, food, delays, seats — without any machine learning.

In [76]:
```
sample = df.sample(1000, random_state=42)
all_words = sample.Review.str.lower().str.split().sum()

pd.Series(all_words).value_counts().head(20)[::-1].plot.barh(color='orange');
```

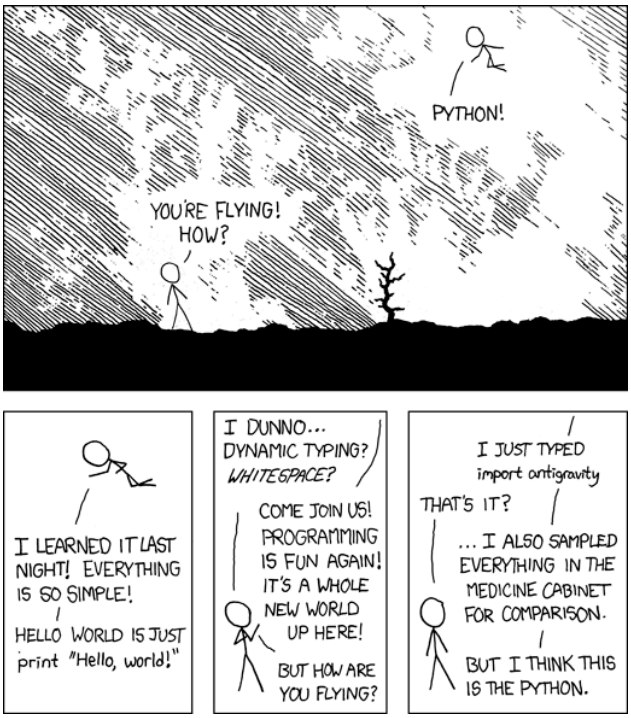


The top terms are mostly **stopwords** — words that carry no meaning (the, a, I, was ...). We can filter them out using a standard list from NLTK:

In [77]:
```
#! pip install nltk
import nltk
nltk.download('stopwords', quiet=True)
from nltk.corpus import stopwords

stop = set(stopwords.words('english'))

filtered_words = [w for w in all_words if w not in stop]
pd.Series(filtered_words).value_counts().head(20)[::-1].plot.barh(color='orange');
```

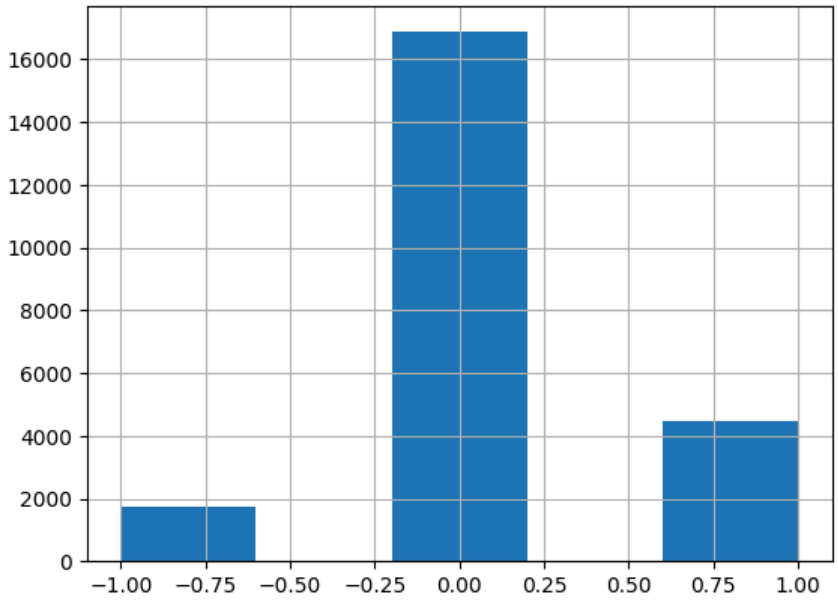


Let's try to plot this as a wordcloud. The beauty and power of python is that - regardless of your objective - you can always find some package that will get close to what you want:



In [78]:
```
#! pip install wordcloud
from wordcloud import WordCloud
import matplotlib.pyplot as plt

text = ' '.join(filtered_words)
wordcloud = WordCloud(width=1200, height=400, background_color='#222222', colormap='summer').generate(text)

plt.figure(figsize=(15, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

flight
one
seat
airline
passenger
service
time
plane
airport
luggage
ticket
staff
hour
bag
good
check
told
food
even
aircraft
customer
delay
said
u
crew
fly
people
way
took
first
got
back
help
asked
day
boarding
terminal
cabin
crew
provided
better
air
price
offered
received
snack
new
later
called
issue
travel
available
lounge
work
well
take
flying
paid
line
still
delayed
flight
attendant
experience
make
staff
never
company
refund
drink
served
another
flew
business
class
board
tried
found
choice
money
website
around
really
two
booked
gate
bad
going
problem
free
quite
thing
wait
given
left
extra
connecting
flight
use
change
overall
minute
made
anything
changed
hotel
poor
best
clean
arrived
counter
without
pay
call
due
booking
bangkok
book
baggage
went
worst
information
return
flight
leg
late
meal
much
go
say
used
family
cancelled
want
enough
put
phone
finally
business
comfortable
nice
little
year
small
water
leaving
reason
waiting
online
option
full
care
everyone
point
flight delayed
old
home
cost
need
via
help
long
find
trip
nothing
back
great
agent
early
lot
customer
excellent
checked
departure
almost
know
able
friendly
email
see
ask
already
terminal
arrival
landing
short
rude
everything
didn t
worst airline
charge
getting
number
bit
month
fee
always
offer
give
came
flown
least
overall
connecting flight
extra
next



Litmus test

This all feels a bit ad-hoc and we should probably validate this with the recommendations to make sure the sentiment makes sense ... Does our simple sentiment score actually correlate with whether passengers recommend the airline?

In [87]:
```
df[['simple_sentiment','Recommended']].corr()
```

Out[87]:

	simple_sentiment	Recommended
simple_sentiment	1.000000	0.319901
Recommended	0.319901	1.000000

Approach 2: Lexicon-Based (VADER)

The **VADER** (Valence Aware Dictionary and sEntiment Reasoner) lexicon was developed by Hutto & Gilbert (2014) at Georgia Tech. They had human annotators rate thousands of words for sentiment intensity, then built a dictionary capturing both polarity (positive/negative) and strength. It handles informal text, punctuation emphasis (!), and capitalization well.

In [82]:
```
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer

nltk.download('vader_lexicon', quiet=True)
sia = SentimentIntensityAnalyzer()
```

In [83]:
```
# Peek at the lexicon — each word has a valence score
print(sia.lexicon.get("amazing"))
print(sia.lexicon.get("terrible"))
print(sia.lexicon.get("okay"))
```

```
2.8
-2.1
0.9
```

In [84]:
```
# polarity_scores returns neg/neu/pos/compound — compound is the overall score (-1 to +1)
sia.polarity_scores('I really loved the snacks on the flight!')
```

Out[84]:
```
{'neg': 0.0, 'neu': 0.572, 'pos': 0.428, 'compound': 0.6689}
```

In [85]:
```
from tqdm import tqdm
tqdm.pandas()

def get_sentiment(review):
    return sia.polarity_scores(review)['compound']

df['sentiment'] = df['Review'].progress_apply(get_sentiment)
```

```
100%|██████████| 23046/23046 [00:12<00:00, 1869.14it/s]
```

You try it

1. Show the 2 most positive and 2 most negative reviews according to VADER.
2. Run the correlation test: does VADER sentiment correlate better with `Recommended` than the ad-hoc approach?

Hint: use `df.sort_values()`

In [ ]:

```

```

Approach 3: Machine Learning

A popular family of machine learning approaches to sentiment analysis is BERT (Bidirectional Encoder Representations from Transformers). Once you know how to use them, they become the swiss army knife of text analytics. BERT models have been fine-tuned for tasks like:

- Sentiment analysis
- Text classification
- Topic detection
- ...

For now, let's keep it simple with a lightweight version of BERT, already fine-tuned for sentiment: `distilbert-base-uncased-finetuned-sst-2-english`. It's fast, free to use, and good enough to show the power of transformer models over rule-based methods.

Note: this may not run on all computers, if it doesn't on yours, switch to google collab.

In [ ]:

```
#! pip install transformers torch
from transformers import pipeline

sentiment_model = pipeline(
    "sentiment-analysis",
    model="distilbert-base-uncased-finetuned-sst-2-english",
    truncation=True,
    max_length=512
)

def scored_sentiment(text):
    result = sentiment_model(text)[0]
    return result['score'] if result['label'] == 'POSITIVE' else -result['score']

# Quick demo
print(scored_sentiment('Best. Flight. Ever.'))
print(scored_sentiment('Worst experience of my life. Never again.'))
```

You try it

Use the `.progress_apply` function to apply sentiment scoring to all BERT sentiments.

In [ ]:

```

```

Going further: emotion detection

Binary positive/negative is useful but coarse. The model below - trained by your professor ;-) on the GoEmotions dataset - classifies text into 28 fine-grained emotions (joy, anger, disappointment, admiration ...). It's based on ModernBERT, a post-ChatGPT architecture, and available on HuggingFace.

In [ ]:

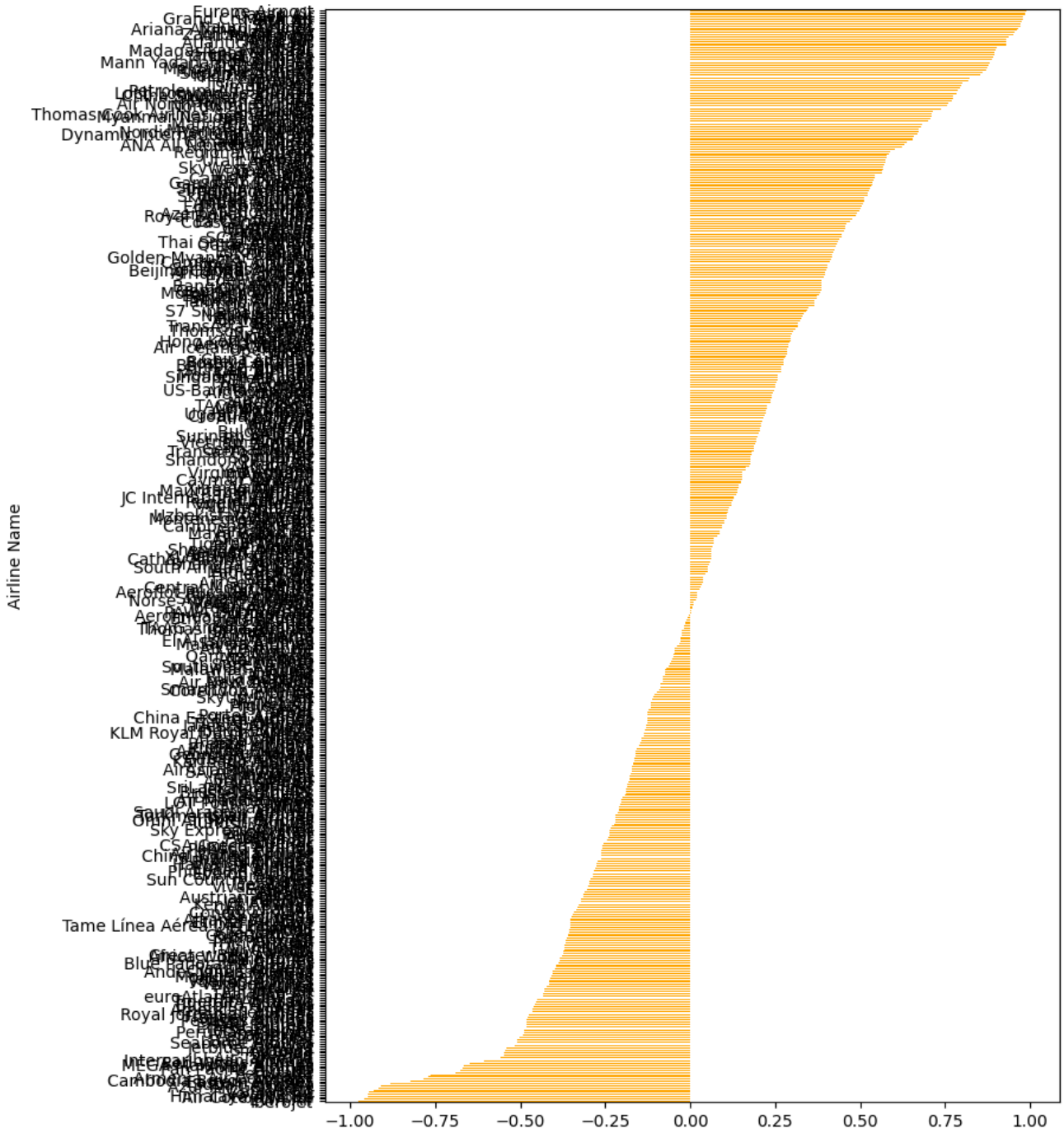
```
from transformers import pipeline

classifier = pipeline(
    "text-classification",
    model           = "cirimus/modernbert-base-go-emotions",
    top_k           = 5
)

text = "The crew was stellar but my why mist my seat be so incredibly uncomfortable for a 12-hour flight."
predictions = classifier(text)

top_5 = sorted(predictions[0], key = lambda x: x["score"], reverse = True)[:5]
print("Top 5 emotions detected:")
for pred in top_5:
    print(f"  {pred['label']:15s}: {pred['score']:.3f}")
```

Who Are the Stars in Star Alliance?

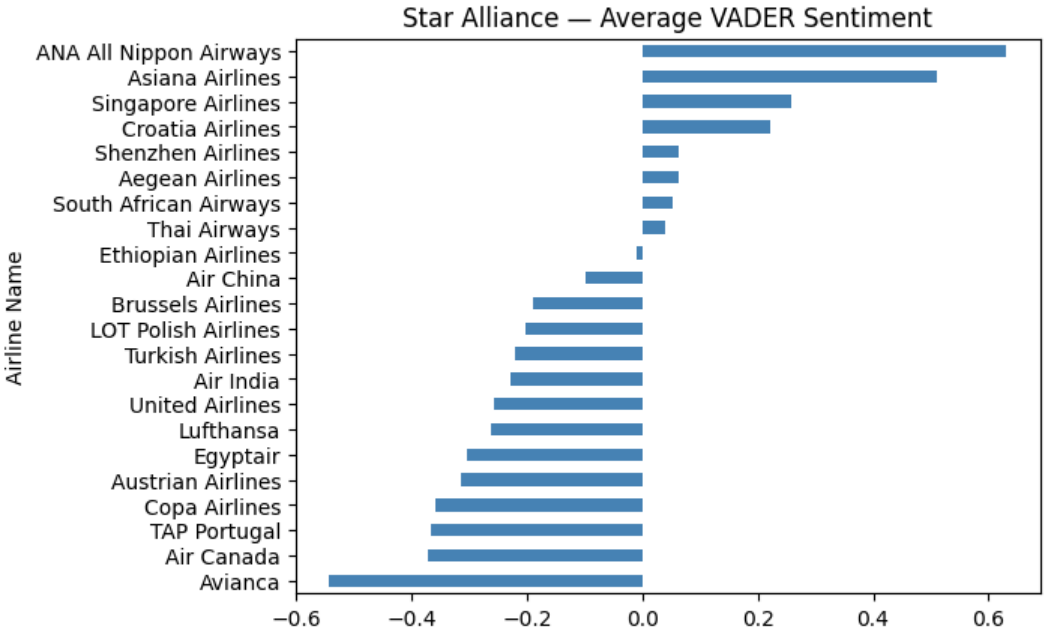
Now that we have sentiment scores, let's rank airlines. We'll filter to Star Alliance members for a focused comparison.

In [50]:

```
df.groupby('Airline Name')['sentiment'].mean().sort_values().plot.barh(
    color='orange', figsize=(8,12)
);
```



In [51]:
```
star_alliance_members = [
    'Aegean Airlines','Air Canada','Air China','Air India',
    'ANA All Nippon Airways','Asiana Airlines','Austrian Airlines',
    'Avianca','Brussels Airlines','Copa Airlines','Croatia Airlines',
    'Egyptair','Ethiopian Airlines','Eva Air','LOT Polish Airlines',
    'Lufthansa','Scandinavian Airlines','Shenzhen Airlines',
    'Singapore Airlines','South African Airways',
    'Swiss International Air Lines','TAP Portugal',
    'Thai Airways','Turkish Airlines','United Airlines'
]

f_star = df['Airline Name'].isin(star_alliance_members)
df[f_star].groupby('Airline Name')['sentiment'].mean().sort_values().plot.barh(
    color='steelblue', title='Star Alliance — Average VADER Sentiment'
);
```



> What follow-up analysis would you do if you worked at Star Alliance?

Part III: Aspect-Based Analysis with LLMs

Overall sentiment tells you whether a review is positive or negative. But an airline category manager needs sharper answers: *How is the crew rated? What do passengers say about luggage? How does food score on long-haul routes?*

This is **aspect-based sentiment analysis** — and it's where LLMs shine. Rather than training a separate model for each aspect (crew, food, luggage, punctuality ...), you just change the prompt.

You have two options for any custom NLP task:

1. Fine-tune your own model (requires labelled data — take the ML class)
2. **Use an LLM to annotate your data** — powerful, fast, and requires no training data

Let's try the second with Google Gemini.

Setup:

1. Create a Google account or log in.
2. Create a free Gemini API key.
3. Install the SDK: `pip install -q -U google-genai`

In [57]:
```
API_KEY = '...'  # paste your key here
```

In [58]:
```
# ! pip install -q -U google-genai
from google import genai

client = genai.Client(api_key=API_KEY)

response = client.models.generate_content(
    model="gemini-2.5-flash",
    contents="Explain in one number what the meaning of life is."
)
print(response.text)
```

```
**42**

(As calculated by the supercomputer Deep Thought in Douglas Adams's *The Hitchhiker's Guide to the Galaxy*, f
or "the Ultimate Question of Life, the Universe, and Everything.")
```

Scoring an aspect: crew

We'll start by scoring the **crew** aspect — one of the most frequently mentioned in airline reviews. The key idea: write a prompt that extracts a structured score from free text. First test your prompt manually in your favourite LLM chat interface, then automate it below.

In [59]:
```
df_ana = df[df['Airline Name'] == 'ANA All Nippon Airways']
df_ana[['Airline Name','Review']].head()
```

Out[59]:

	Airline Name	Review
4350	ANA All Nippon Airways	Pretty disappointed with ANA, especially when compared to JAL. Seats are dated and half are very exposed. The Japanese food was generally good and at times excellent. The champagne was undrinkable. The pre-flight drinks (where offered) were from plastic disposable cups. The main listed wine selection on every leg was unavailable. The crew were generally good and attentive. It is such a pity that they appear to be economising.
4351	ANA All Nippon Airways	Great service, lovely crew, comfortable seats, lots of leg room (I am 6'2"), roomy toilets and an immaculately clean aircraft. It's hard to imagine how a 14 hour flight in economy could be done any better. I paid in advance an extra £15 for a special Japanese meal which was most definitely worth it and really delicious, beautifully presented.
4352	ANA All Nippon Airways	Overall was a good experience, check in went smoothly, food was decent for airline food and the staff was friendly. The main downside was that the lounge in Tokyo Haneda was overcrowded.
4353	ANA All Nippon Airways	Paid for seats so entire family is together. Airline changes aircraft/seats and puts a stranger between us. Not even a single thought goes in this process. Extremely difficult to connect with someone on the phone. Unfortunately the competition is worse. Over the years flying changed from a fun way to travel to a dreadful experience.
4354	ANA All Nippon Airways	I'm super unhappy about ANA service. They asked me and my daughter to move our luggage after we have settled down, me and my daughter are both short, but she didn't offer her help, so that I almost dropped my luggage on one of passengers. Later during the breakfast time, she didn't offer food to my daughter until I figured it out. My daughter is shy girl, so I had to step out and ask her to bring the breakfast to her. This kind of things have never happened to me before. I'm extremely unhappy about ANA service.

In [61]:
```
PROMPT = """
You're a world-class customer service analyst who is analyzing airline reviews.
Score the crew on the flight (friendliness, professionalism, helpfulness) from 0 to 10.
If crew is not mentioned, return 'N/A'.

Also extract the most relevant passage that supports your score.

Format your response as:

<score>YOUR_SCORE_HERE</score><passage>YOUR_PASSAGE_HERE</passage>

Important: only reply with the required tags and scores. Don't add anything else.
"""

def score_crew(review_text):
    prompt = PROMPT + "\nReview:" + review_text
    response = client.models.generate_content(
        model="gemini-2.5-flash",
        contents=prompt
    )
    return response.text.strip()

score_crew('the crew was amazing. the food was good too.')
```

Out[61]:
```
'<score>10</score><passage>the crew was amazing.</passage>'
```

You try it

1. Apply `score_crew()` to `df_sample` below (25 rows - don't run on the full dataset yet, it uses API quota).
2. Parse the output: create a `CrewScore` column and a `Rationale` column.
3. **Bonus:** change the `PROMPT` to score a *different* aspect - luggage handling, food quality, or punctuality. How little do you need to change to pivot to a completely different analysis?

Hints:

- Use `.apply(score_crew)` or `.progress_apply(score_crew)` to get a column of raw LLM output strings.
- Use string splitting on `<score>` and `<passage>` tags to extract the values.
- LLMs sometimes don't follow instructions ... filter out rows where the tags are missing.

In [ ]:
```
df_sample = df.sample(25, random_state=42)
# llm_scores = df_sample['Review'].apply(score_crew)
# ...
```