AirTrack

December 29, 2025

[1]:

```python
import requests

LAT = 45.07
LON = 7.69
RADIUS_NM = 150  # max 250 nm

url = f"https://api.airplanes.live/v2/point/{LAT}/{LON}/{RADIUS_NM}"
r = requests.get(url, timeout=30)
print("Status:", r.status_code)
r.raise_for_status()

data = r.json()
type(data), list(data.keys())[:10]
```

```
Status: 200
```

[1]:

```
(dict, ['ac', 'msg', 'now', 'total', 'ctime', 'ptime'])
```

[2]:

```python
import pandas as pd
from datetime import datetime, timezone

aircraft = data.get("ac", [])
print("Aircraft records:", len(aircraft))

df = pd.DataFrame(aircraft)
df["snapshot_time_utc"] = datetime.now(timezone.utc).isoformat()

df.head()
```

```
Aircraft records: 230
```

[2]:

```
      hex       type    flight       r     t                       desc  \
0  502d5d  adsb_icao  TVF52LZ   YL-ABL  BCS3             AIRBUS A220-300
1  39a123  adsb_icao  IXR958    F-HIJD  C25A  CESSNA 525A Citation CJ2
2  452166  adsb_icao  DAH1010   LZ-FSF  A320                AIRBUS A-320
3  3986e9  adsb_icao  AFR53BQ   F-HBXJ  E170          EMBRAER ERJ-170-100
4  39bda2  adsb_icao  AFR69VB   F-HPNC  BCS3             AIRBUS A220-300

  alt_baro  alt_geom     gs     ias  …  rssi      dst     dir  \
0    30000   30450.0  442.4  272.0  … -10.5  148.017  288.0
1    23625   24050.0  350.2  270.0  … -15.9  142.428  271.6
2    34000   34350.0  456.0  259.0  …  -4.9  145.393  291.3
3    11925   12425.0  330.1  276.0  …  -9.4  141.256  286.6
4    22700   23150.0  389.8  310.0  … -18.7  144.856  252.9

   nav_altitude_fms  emergency          nav_modes  dbFlags  ownOp  year  \
0               NaN        NaN                NaN      NaN    NaN   NaN
1            4208.0        NaN                NaN      NaN    NaN   NaN
2               NaN       none                NaN      NaN    NaN   NaN
3            7008.0        NaN  [autopilot, tcas]      NaN    NaN   NaN
4           19008.0        NaN                NaN      NaN    NaN   NaN

                  snapshot_time_utc
0  2025-12-29T10:23:33.435470+00:00
1  2025-12-29T10:23:33.435470+00:00
2  2025-12-29T10:23:33.435470+00:00
3  2025-12-29T10:23:33.435470+00:00
4  2025-12-29T10:23:33.435470+00:00

[5 rows x 57 columns]
```

[3]:

```python
out = "airplanes_live_snapshot.csv"
df.to_csv(out, index=False)
print("Saved:", out)
```

```
Saved: airplanes_live_snapshot.csv
```

[4]:

```python
import time

snapshots = []
N = 60          # 60 snapshots
SLEEP = 1.1     # >1 sec to respect the limit

for i in range(N):
    r = requests.get(url, timeout=30)
    r.raise_for_status()
    payload = r.json()

    dfi = pd.DataFrame(payload.get("ac", []))
    dfi["snapshot_time_utc"] = datetime.now(timezone.utc).isoformat()
    snapshots.append(dfi)

    print(f"{i+1}/{N}: rows={len(dfi)}")
    time.sleep(SLEEP)

df_all = pd.concat(snapshots, ignore_index=True)
df_all.to_csv("airplanes_live_timeseries.csv", index=False)
```

```python
print("Saved: airplanes_live_timeseries.csv rows=", len(df_all))
```

```
1/60: rows=229
2/60: rows=230
3/60: rows=229
4/60: rows=230
5/60: rows=230
6/60: rows=230
7/60: rows=230
8/60: rows=229
9/60: rows=230
10/60: rows=230
11/60: rows=229
12/60: rows=229
13/60: rows=229
14/60: rows=229
15/60: rows=227
16/60: rows=227
17/60: rows=227
18/60: rows=227
19/60: rows=227
20/60: rows=226
21/60: rows=227
22/60: rows=226
23/60: rows=226
24/60: rows=227
25/60: rows=226
26/60: rows=226
27/60: rows=226
28/60: rows=227
29/60: rows=227
30/60: rows=226
31/60: rows=226
32/60: rows=225
33/60: rows=225
34/60: rows=225
35/60: rows=225
36/60: rows=225
37/60: rows=224
38/60: rows=223
39/60: rows=223
40/60: rows=223
41/60: rows=223
42/60: rows=222
43/60: rows=221
44/60: rows=220
45/60: rows=221
46/60: rows=221
```

```
47/60: rows=221
48/60: rows=220
49/60: rows=219
50/60: rows=219
51/60: rows=219
52/60: rows=221
53/60: rows=223
54/60: rows=223
55/60: rows=224
56/60: rows=223
57/60: rows=223
58/60: rows=221
59/60: rows=221
60/60: rows=221
Saved: airplanes_live_timeseries.csv rows= 13508
```

[5]:

```python
from google.colab import files
files.download("airplanes_live_snapshot.csv")
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

[6]:

```python
import pandas as pd
import numpy as np
import pandas as pd

df = df.copy()
print(df.shape)
```

```
(230, 57)
```

[7]:

```python
df["snapshot_time_utc"] = pd.to_datetime(df["snapshot_time_utc"], utc=True,␣
 ↪errors="coerce")

# Normalize common string fields
for c in ["hex", "type", "flight", "r", "t", "desc", "category", "squawk",␣
 ↪"emergency"]:
    if c in df.columns:
        df[c] = df[c].astype("string")

if "flight" in df.columns:
    df["flight"] = df["flight"].str.strip()

# Columns that should be numeric
num_cols = [
    "alt_baro","alt_geom","gs","ias","tas","mach","wd","ws","oat","tat",
    "track","track_rate","roll","mag_heading","true_heading",
    "baro_rate","geom_rate","nav_qnh","nav_altitude_mcp","nav_heading",
    "lat","lon","nic","rc","seen_pos","version","nic_baro","nac_p","nac_v",
    "sil","gva","sda","alert","spi","mlat","tisb","messages","seen","rssi",
    "dst","dir","nav_altitude_fms","dbFlags","ownOp","year"
]
for c in num_cols:
    if c in df.columns:
        df[c] = pd.to_numeric(df[c], errors="coerce")

# Wrap angles to [0, 360)
for c in ["track","mag_heading","true_heading","wd","dir","nav_heading"]:
    if c in df.columns:
        df[c] = df[c] % 360

df.shape
```

[7]:

```
(230, 57)
```

[8]:

```
def pct(x):
    return round(100 * x, 2)

summary = {}
summary["rows"] = len(df)
summary["unique_hex"] = df["hex"].nunique(dropna=True) if "hex" in df.columns␣
  ↪else None
summary["unique_flights"] = df["flight"].nunique(dropna=True) if "flight" in df.
  ↪columns else None
summary["time_min"] = df["snapshot_time_utc"].min()
summary["time_max"] = df["snapshot_time_utc"].max()

# Emergency flags / squawk presence
if "emergency" in df.columns:
    summary["emergency_nonempty"] = int(df["emergency"].fillna("").str.len().
  ↪gt(0).sum())
if "squawk" in df.columns:
    summary["squawk_nonnull"] = int(df["squawk"].notna().sum())

summary
```

[8]:

```
{'rows': 230,
 'unique_hex': 230,
 'unique_flights': 229,
 'time_min': Timestamp('2025-12-29 10:23:33.435470+0000', tz='UTC'),
 'time_max': Timestamp('2025-12-29 10:23:33.435470+0000', tz='UTC'),
 'emergency_nonempty': 108,
 'squawk_nonnull': 224}
```

[9]:

```
missing = (df.isna().mean().sort_values(ascending=False) * 100).round(1)
missing.head(25)
```

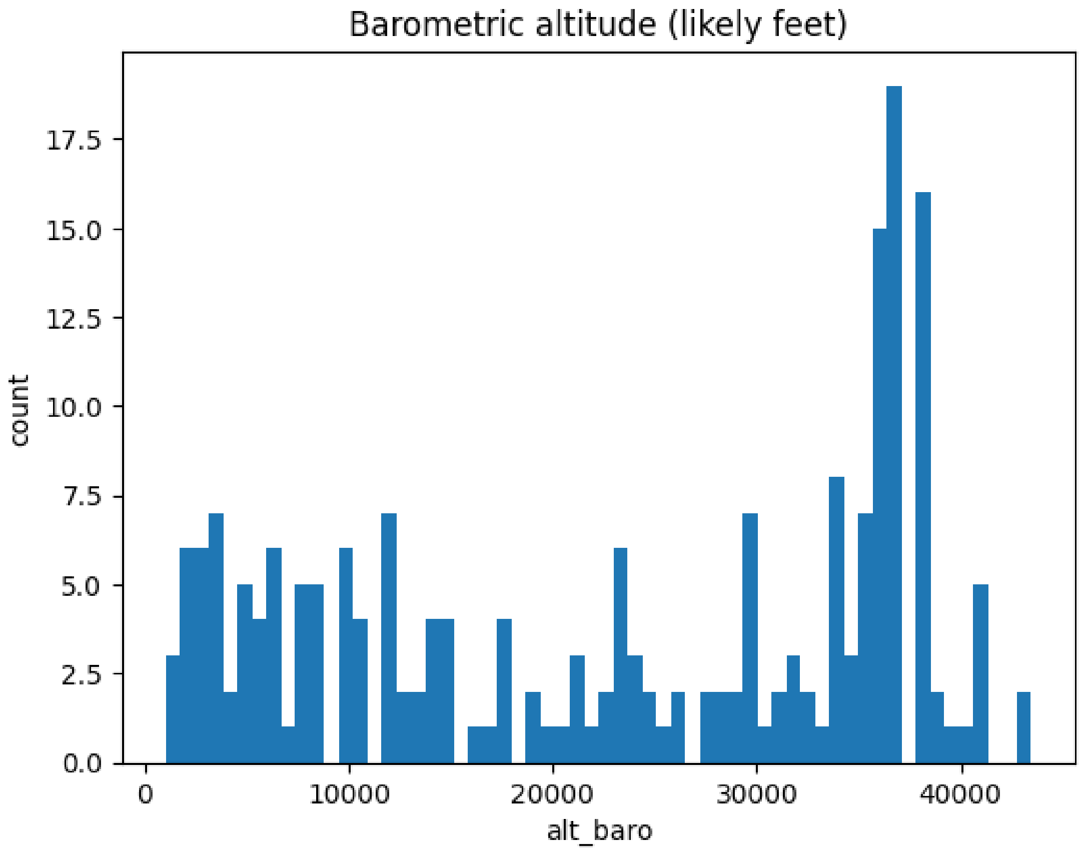
[9]:

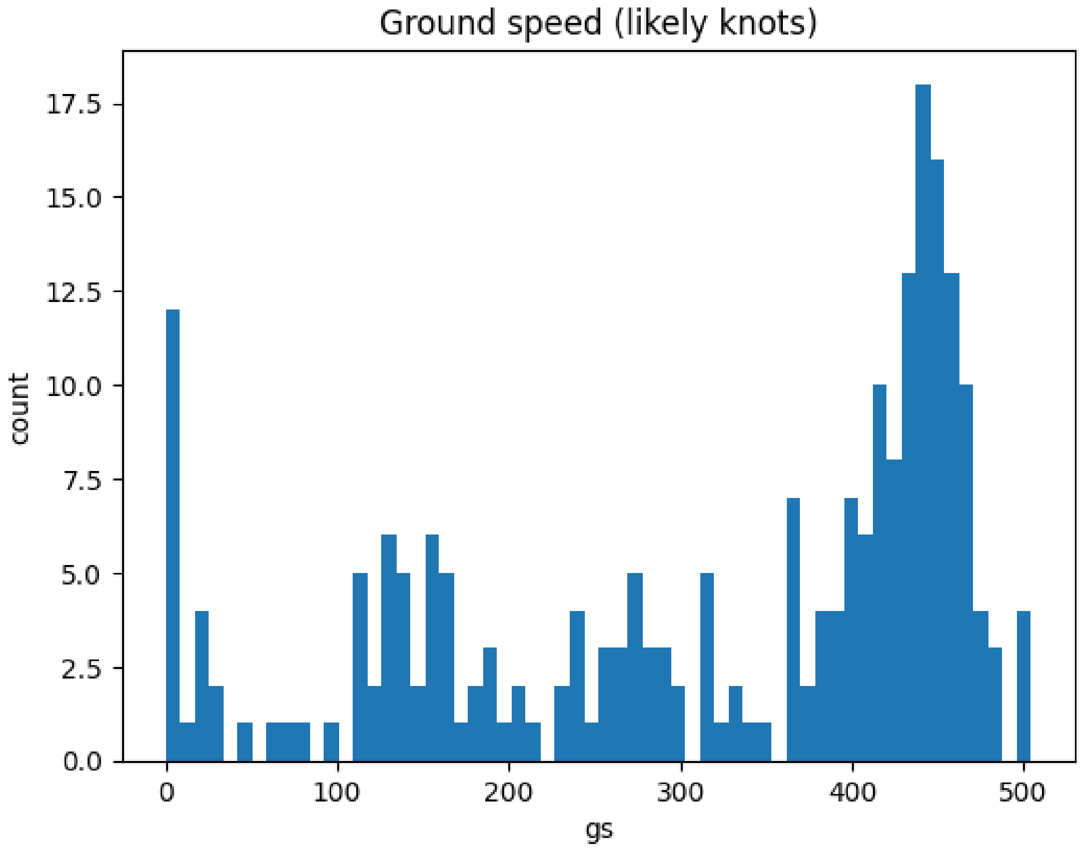
```
mlat                  100.0
ownOp                 100.0
tisb                  100.0
dbFlags                99.1
year                   97.8
nav_modes              91.3
nav_altitude_fms       64.8
emergency              53.0
nav_heading            44.8
oat                    31.3
tat                    31.3
track_rate             26.1
wd                     23.9
ws                     23.9
roll                   20.9
tas                    20.4
geom_rate              20.0
mach                   19.1
ias                    19.1
mag_heading            18.3
nav_altitude_mcp       14.8
true_heading           13.9
nav_qnh                13.0
gva                    11.3
alt_geom                9.6
dtype: float64
```

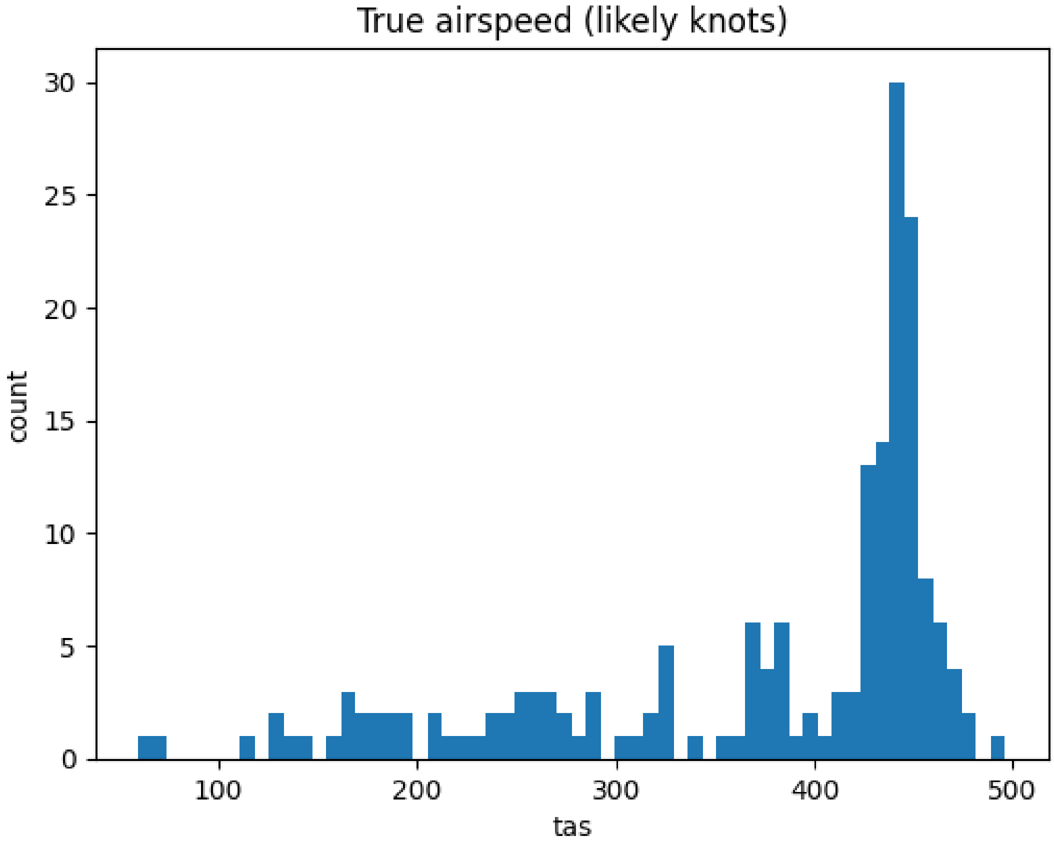
[10]:

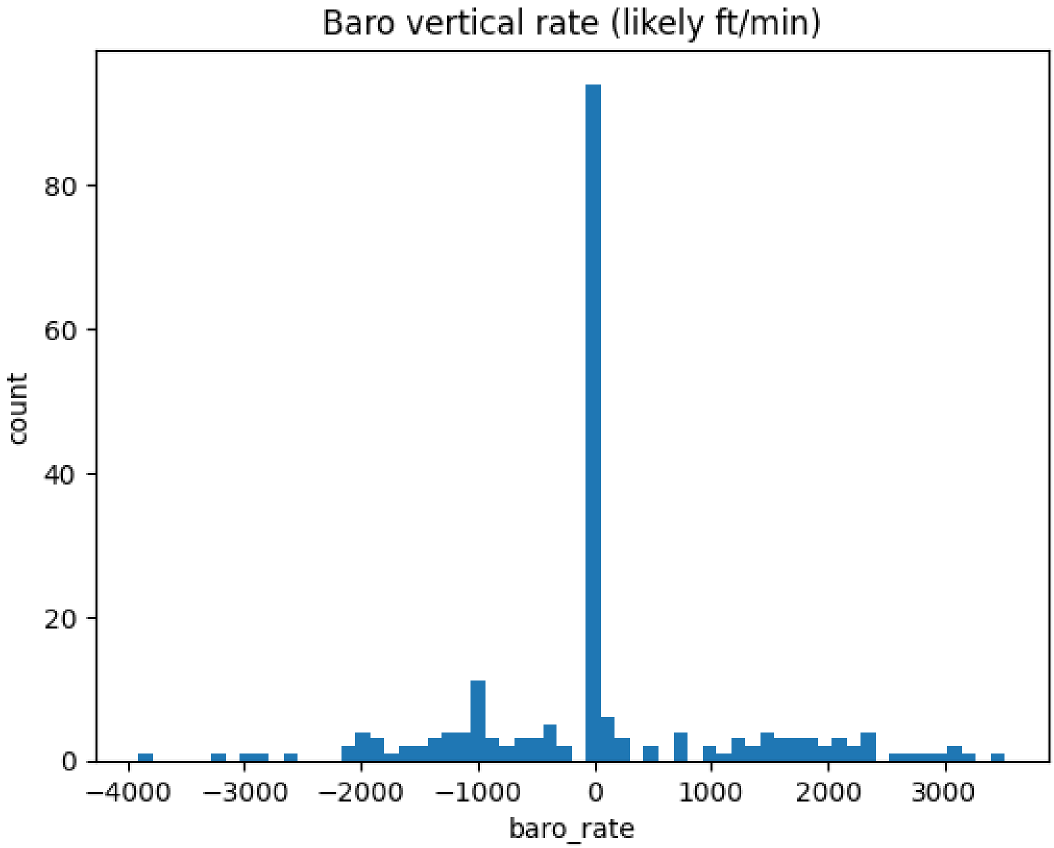
```
import matplotlib.pyplot as plt

def hist(col, title, bins=60):
    if col not in df.columns:
        print(f"Missing column: {col}")
        return
    s = df[col].dropna()
    if s.empty:
        print(f"No data for: {col}")
        return
    plt.figure()
    plt.hist(s, bins=bins)
    plt.title(title)
    plt.xlabel(col)
    plt.ylabel("count")
    plt.show()
```

```
hist("alt_baro", "Barometric altitude (likely feet)")
hist("gs", "Ground speed (likely knots)")
hist("tas", "True airspeed (likely knots)")
hist("baro_rate", "Baro vertical rate (likely ft/min)")
```



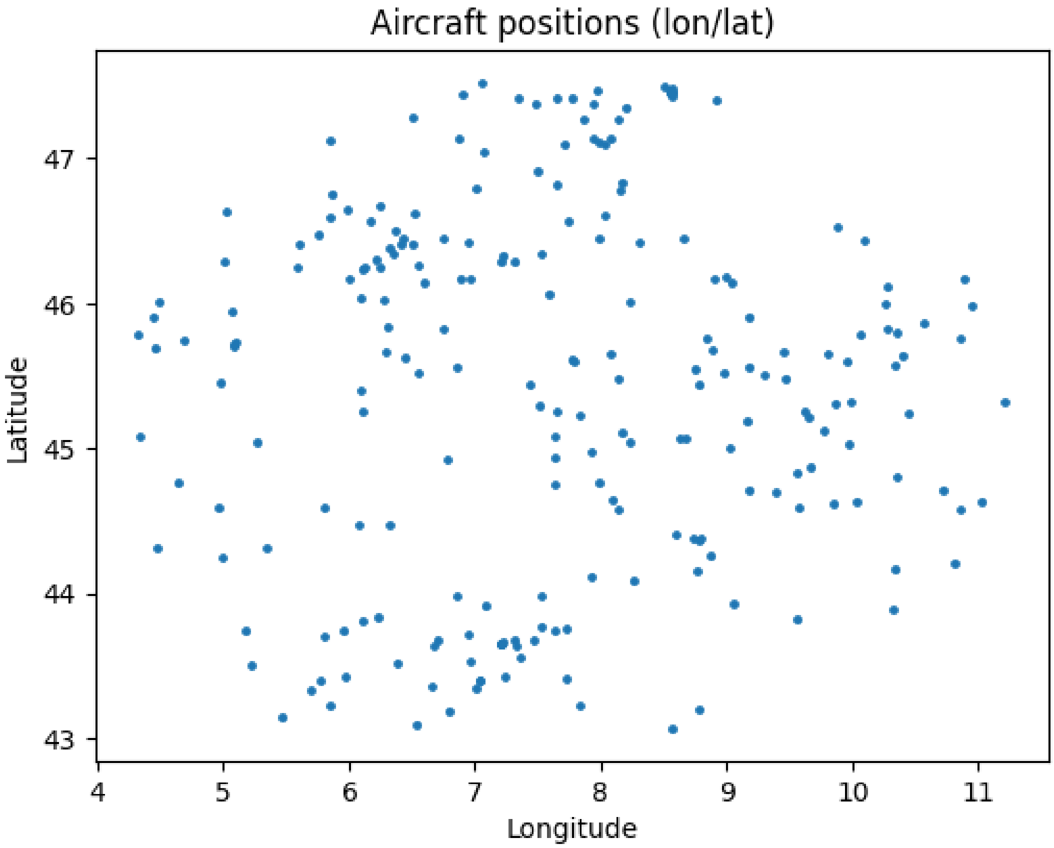

Ground speed (likely knots)
count
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
100
200
300
400
500
gs


True airspeed (likely knots)
count
0
5
10
15
20
25
30
100
200
300
400
500
tas



[11]:

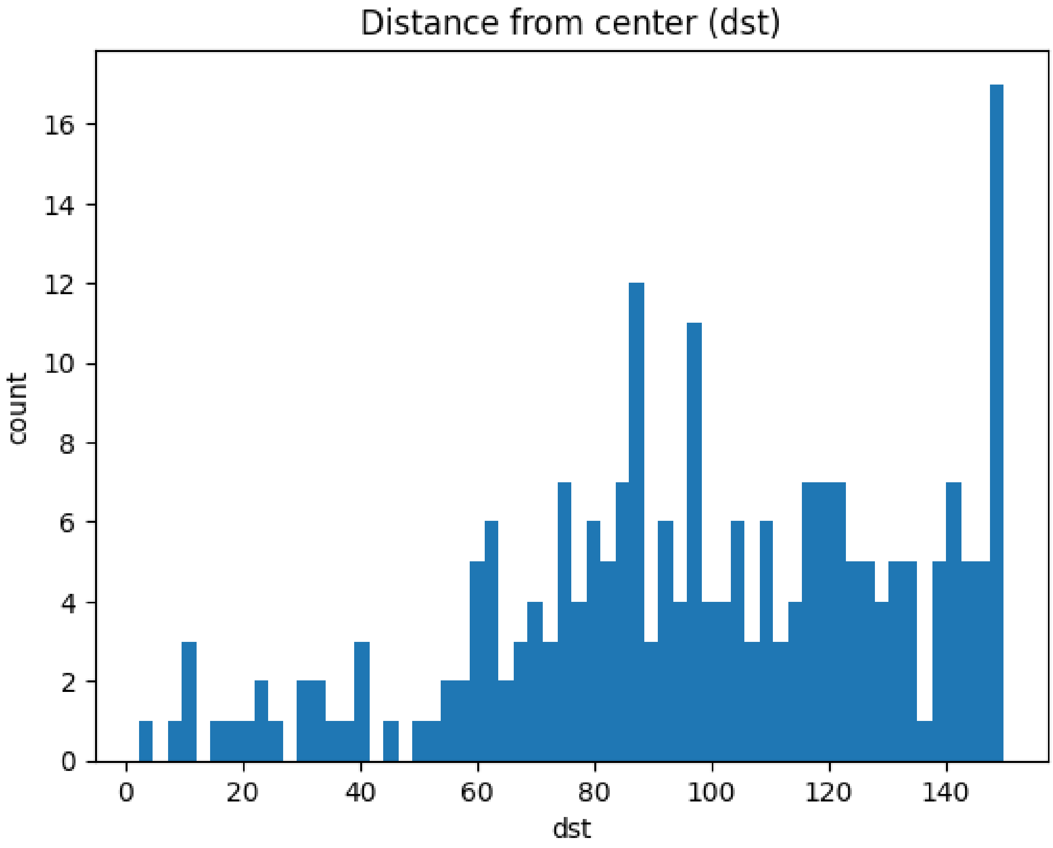
```
import matplotlib.pyplot as plt

if {"lat","lon"}.issubset(df.columns):
    d = df[["lat","lon"]].dropna()
    plt.figure()
    plt.scatter(d["lon"], d["lat"], s=6)
    plt.title("Aircraft positions (lon/lat)")
    plt.xlabel("Longitude")
    plt.ylabel("Latitude")
    plt.show()
else:
    print("lat/lon not available")
```

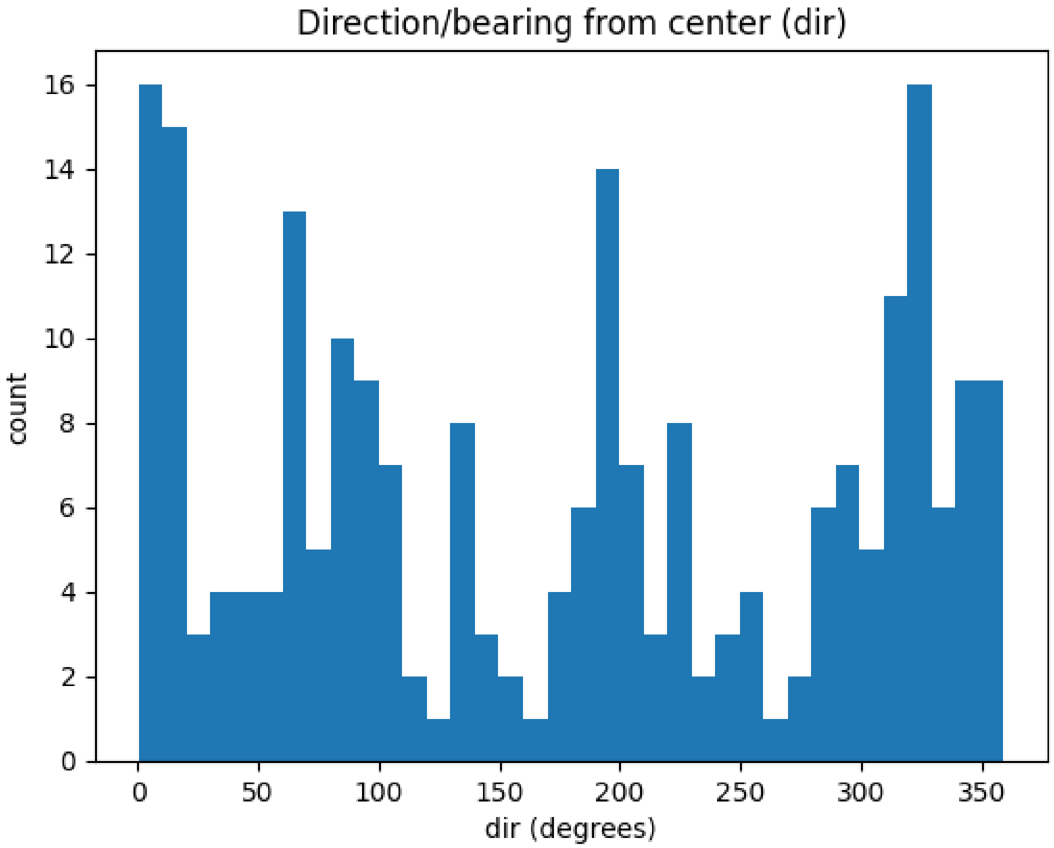


[12]:
```
if "dst" in df.columns:
    print("dst (distance) summary:")
    display(df["dst"].describe())

    plt.figure()
    plt.hist(df["dst"].dropna(), bins=60)
    plt.title("Distance from center (dst)")
    plt.xlabel("dst")
    plt.ylabel("count")
    plt.show()

if "dir" in df.columns:
    plt.figure()
    plt.hist(df["dir"].dropna(), bins=36)
    plt.title("Direction/bearing from center (dir)")
    plt.xlabel("dir (degrees)")
    plt.ylabel("count")
    plt.show()
```

```
dst (distance) summary:
```

```
count    230.000000
mean      98.285591
std       35.273824
min        2.251000
25%       76.649750
50%       97.913500
75%      125.782000
max      149.949000
Name: dst, dtype: float64
```





[13]:

```
def classify_phase(vr_fpm):
    if pd.isna(vr_fpm):
        return "unknown"
    if vr_fpm > 300:
        return "climb"
    if vr_fpm < -300:
        return "descent"
    return "level"

df["phase"] = df["baro_rate"].apply(classify_phase) if "baro_rate" in df.
  ↪columns else "unknown"
df["phase"].value_counts(dropna=False)
```

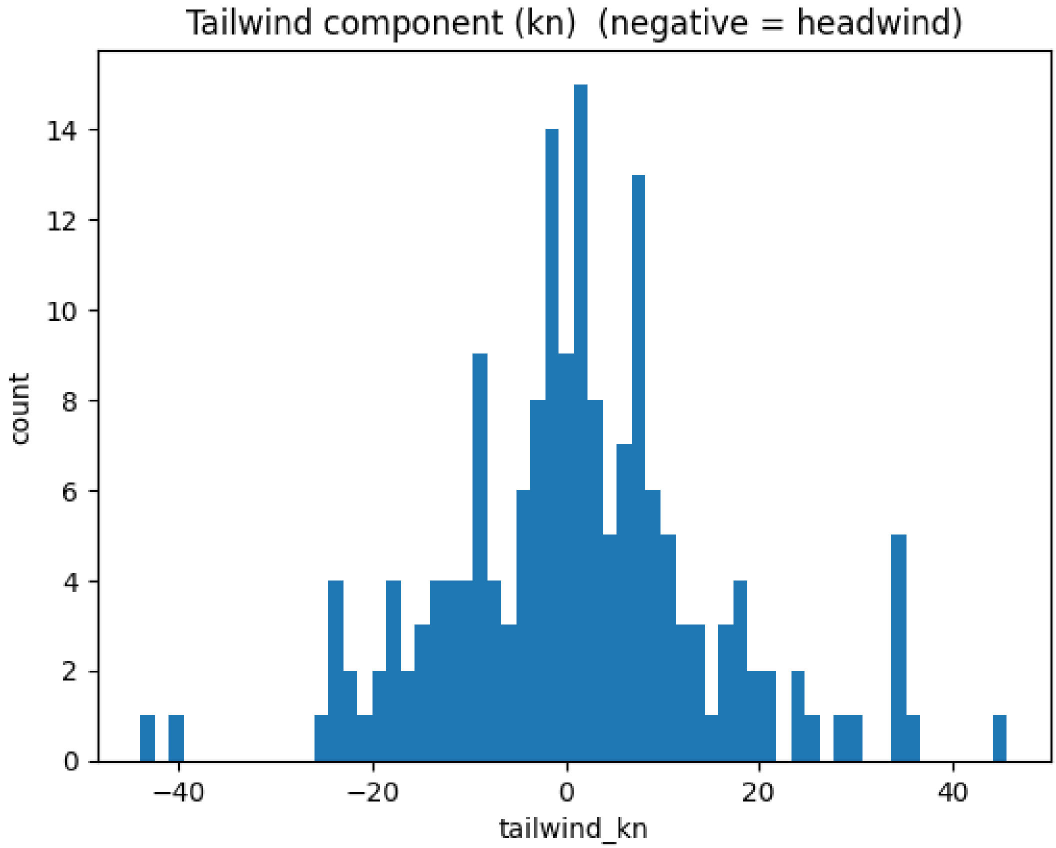
[13]:

```
phase
level      105
descent     57
climb       46
unknown     22
Name: count, dtype: int64
```

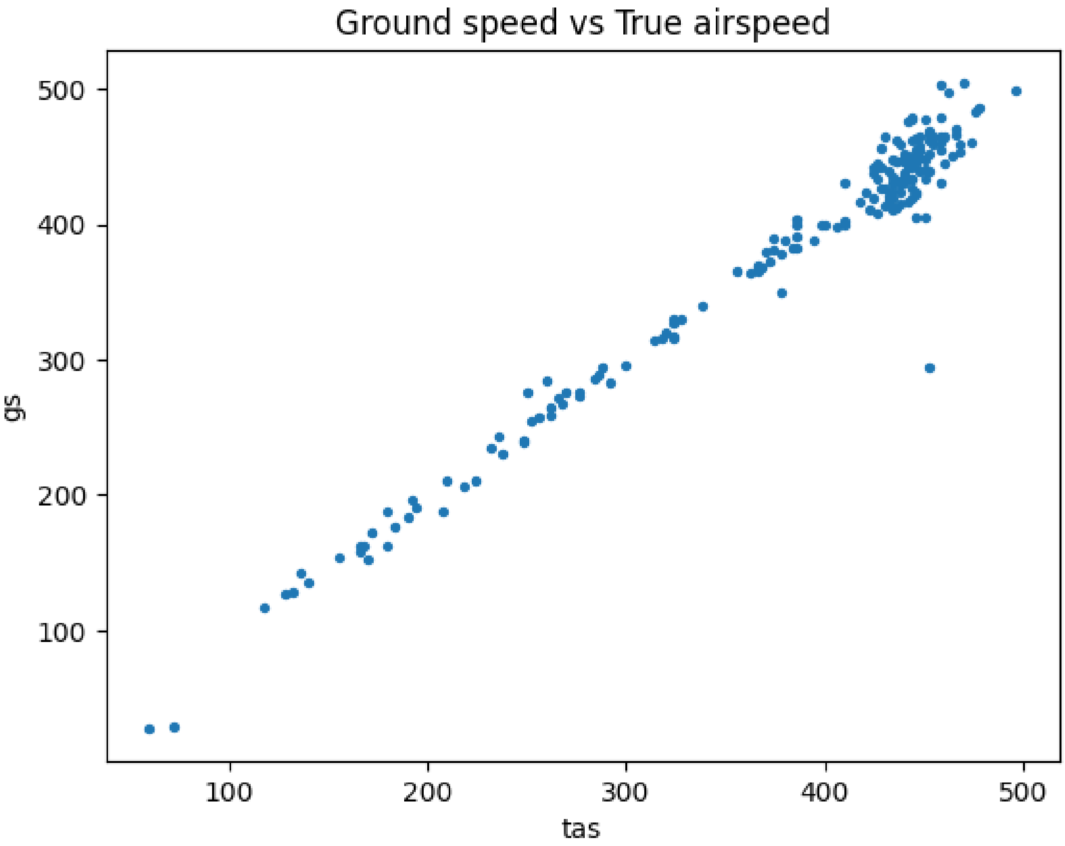
[14]:

```
import numpy as np

needed = {"wd","ws","track"}
if needed.issubset(df.columns):
    # Convert degrees to radians
    trk = np.deg2rad(df["track"])
    wd = np.deg2rad(df["wd"])  # wind direction (usually "from", meteorological␣
  ↪convention)

    ws = df["ws"]

    # If wd is "from", the wind vector points opposite direction of wd.
    # Headwind positive when opposing motion along track.
    # We'll compute components along the aircraft track axis.
    wind_to = wd + np.pi  # direction wind is going TO

    headwind = ws * np.cos(wind_to - trk)       # + = tailwind, - = headwind␣
  ↪(by this sign convention)
    crosswind = ws * np.sin(wind_to - trk)      # signed crosswind

    df["tailwind_kn"] = headwind
    df["crosswind_kn"] = crosswind

    display(df[["ws","wd","track","tailwind_kn","crosswind_kn"]].dropna().
  ↪head(10))

    plt.figure()
    plt.hist(df["tailwind_kn"].dropna(), bins=60)
    plt.title("Tailwind component (kn)  (negative = headwind)")
    plt.xlabel("tailwind_kn")
    plt.ylabel("count")
    plt.show()
else:
    print("Need wd, ws, track columns for wind components.")
```

```
     ws     wd   track  tailwind_kn  crosswind_kn
0  38.0  125.0    4.54    19.263595    -32.755365
1  27.0  131.0  148.31   -25.777140      8.033621
2  45.0  124.0  353.45    29.255012    -34.192752
3   7.0  212.0  113.38     1.049163     -6.920929
4  43.0  120.0    7.96    16.135913    -39.857650
5   7.0  283.0  127.84     6.352391     -2.940600
6  29.0  124.0  355.73    17.961673    -22.767923
7  15.0  157.0  233.98    -3.379367     14.614372
8  44.0  141.0  142.51   -43.984721      1.159463
9  14.0  170.0   51.47     6.686664    -12.299940
```



[15]:
```
if {"gs","tas"}.issubset(df.columns):
    d = df[["gs","tas"]].dropna()
    plt.figure()
    plt.scatter(d["tas"], d["gs"], s=8)
    plt.title("Ground speed vs True airspeed")
    plt.xlabel("tas")
    plt.ylabel("gs")
    plt.show()

    df["gs_minus_tas"] = df["gs"] - df["tas"]
    display(df["gs_minus_tas"].describe())
else:
    print("Need gs and tas.")
```



```
count    183.000000
mean      -0.668306
std       18.676844
min     -157.800000
25%       -8.200000
50%        0.400000
75%        7.350000
max       45.400000
Name: gs_minus_tas, dtype: float64
```

[16]:
```
if "hex" in df.columns:
    display(df["hex"].value_counts().head(15).to_frame("rows"))

if "messages" in df.columns:
    display(df[["hex","flight","messages"]].dropna().sort_values("messages",␣
  ↪ascending=False).head(15))

if "seen" in df.columns:
    display(df[["hex","flight","seen"]].dropna().sort_values("seen").head(15)) ␣
  ↪# smaller = seen more recently in many feeds
```

```
        rows
hex
502d5d     1
39a123     1
452166     1
3986e9     1
39bda2     1
440cff     1
49d4f7     1
501c16     1
440452     1
3986e5     1
394c10     1
39d313     1
4cadfa     1
440d6f     1
39856d     1
```

```
        hex   flight  messages
229  485a35   KLM85A    376292
145  4078fd   EXS4VS    348112
104  45d20a   JNJ586    338793
12   4cadfa   ITY621    325812
227  486491   KLM27U    314925
58   4408c9  EZS78YX    277607
67   4856ed    PHBRA    271445
18   4ca8d8  RYR74KV    265959
134  485a34  KLM1597    263899
210  ab6851   DAL288    262036
160  4b1698   SWR7LX    249756
221  4892c2   ENT9LK    242797
193  4d226f  RYR5411    239358
44   49d328   TVS3CE    236308
52   4b1a27  EZS45KG    225829
```

```
       hex   flight  seen
1   39a123   IXR958   0.0
2   452166  DAH1010   0.0
5   440cff  EJU96PZ   0.0
12  4cadfa   ITY621   0.0
11  39d313  TVF8223   0.0
10  394c10  AFR41HM   0.0
9   3986e5  AFR6269   0.0
8   440452  EJU62EA   0.0
25  3e255e   MCK307   0.0
27  3b9b60  FNY5012   0.0
30  4b17e3   SWR5TA   0.0
29  4d2224  RYR87TU   0.0
31  4caee5  RYR36BA   0.0
```

```
22  3c66b9   OCN92P   0.0
21  39dd54  CCM64XG   0.0
```

```
WARNING: Runtime no longer has a reference to this dataframe, please re-run this
cell and try again.
```

[17]:
```
!pip -q install reportlab
```

```
   ---------------------------------------- 0.0/2.0 MB
? eta -:--:--
---------------------------------------- 0.3/2.0

MB 9.3 MB/s eta 0:00:01
---------------------------------------- 1.9/2.0 MB
30.7 MB/s eta 0:00:01
---------------------------------------- 2.0/2.0 MB 23.8

MB/s eta 0:00:00
```

[18]:
```
from reportlab.lib.pagesizes import A4, landscape
from reportlab.platypus import SimpleDocTemplate, Table, TableStyle, PageBreak,␣
 ↪Paragraph, Spacer
from reportlab.lib import colors
from reportlab.lib.styles import getSampleStyleSheet

def dataframe_to_pdf(df, filename="aircraft_data.pdf", max_rows=300,␣
 ↪rows_per_page=35):
    # Limit rows for sanity (adjust max_rows if you really want more)
    d = df.head(max_rows).copy()

    # Convert everything to strings to avoid ReportLab type issues
    d = d.fillna("").astype(str)

    styles = getSampleStyleSheet()
    doc = SimpleDocTemplate(filename, pagesize=landscape(A4), rightMargin=18,␣
 ↪leftMargin=18, topMargin=18, bottomMargin=18)

    elements = []
    elements.append(Paragraph("Aircraft Data Export", styles["Title"]))
    elements.append(Spacer(1, 12))
    elements.append(Paragraph(f"Rows exported: {len(d)} (of {len(df)})",␣
 ↪styles["Normal"]))
    elements.append(Spacer(1, 12))

    data = [list(d.columns)] + d.values.tolist()

    # Split into pages
    header = data[0]
    body = data[1:]

    for start in range(0, len(body), rows_per_page):
        chunk = [header] + body[start:start + rows_per_page]

        table = Table(chunk, repeatRows=1)

        table.setStyle(TableStyle([
            ("BACKGROUND", (0,0), (-1,0), colors.lightgrey),
            ("TEXTCOLOR", (0,0), (-1,0), colors.black),
            ("GRID", (0,0), (-1,-1), 0.25, colors.grey),
            ("FONTNAME", (0,0), (-1,0), "Helvetica-Bold"),
            ("FONTSIZE", (0,0), (-1,-1), 7),
            ("VALIGN", (0,0), (-1,-1), "MIDDLE"),
            ("ROWBACKGROUNDS", (0,1), (-1,-1), [colors.whitesmoke, colors.
 ↪white]),
        ]))

        elements.append(table)
        if start + rows_per_page < len(body):
            elements.append(PageBreak())

    doc.build(elements)
    return filename

pdf_name = dataframe_to_pdf(df, filename="airplanes_data.pdf", max_rows=300,␣
 ↪rows_per_page=35)
pdf_name
```

[18]:

```
'airplanes_data.pdf'
```

[19]:

```
from google.colab import files
files.download("airplanes_data.pdf")
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```