Airbnb Project – NYC

This notebook:

- Runs EDA with charts: price distribution, listings by neighbourhood, box plot of price by room type, price vs reviews, correlation heatmap.
- Saves figures and stats to an `outputs/`
- Includes an ML section (linear regression for price) and simple clustering.
- Provides instructions to run the Streamlit dashboard that ships with this project.

In [1]:
```
%pip install -q pandas numpy matplotlib plotly scikit-learn request
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [1]:
```

import os
import io
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import requests

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.cluster import KMeans

# Paths and constants
DATA_URL = "https://data.insideairbnb.com/united-states/ny/new-york
DATA_PATH = "listings.csv"
OUT_DIR = "outputs"
os.makedirs(OUT_DIR, exist_ok=True)

# Inline plots
%matplotlib inline
```

```
/Users/shivalimuthukumar/anaconda3/lib/python3.11/site-packages/pa
ndas/core/arrays/masked.py:61: UserWarning: Pandas requires versio
n '1.3.6' or newer of 'bottleneck' (version '1.3.5' currently inst
alled).
  from pandas.core import (
```

Download the NYC listings.csv

In [2]:
```python
def download_if_missing(url: str, path: str) -> str:
    if os.path.exists(path) and os.path.getsize(path) > 0:
        print(f"Found local {path}. Using it.")
        return path
    print("Downloading:", url)
    r = requests.get(url, timeout=120)
    r.raise_for_status()
    with open(path, "wb") as f:
        f.write(r.content)
    print("Saved to", path)
    return path

_ = download_if_missing(DATA_URL, DATA_PATH)
```

```
Downloading: https://data.insideairbnb.com/united-states/ny/new-yo
rk-city/2025-08-01/visualisations/listings.csv (https://data.insid
eairbnb.com/united-states/ny/new-york-city/2025-08-01/visualisatio
ns/listings.csv)
Saved to listings.csv
```

Load and clean data

In [3]:
```python
def coerce_price(series: pd.Series) -> pd.Series:
    try:
        return (
            series.astype(str)
            .str.replace("$", "", regex=False)
            .str.replace(",", "", regex=False)
            .str.replace(" ", "", regex=False)
            .str.extract(r'([0-9]*\.?[0-9]+)')[0]
            .astype(float)
        )
    except Exception:
        return pd.to_numeric(series, errors="coerce")

def basic_clean(df: pd.DataFrame) -> pd.DataFrame:
    df = df.drop_duplicates().copy()
    if "price" in df.columns:
        df["price"] = coerce_price(df["price"])
    # Standard numeric casting and missing handling
    for c in df.columns:
        if df[c].dtype.kind in "biufc":
            if df[c].isna().mean() < 0.95:
                df[c] = pd.to_numeric(df[c], errors="coerce")
                df[c] = df[c].fillna(df[c].median())
        else:
            df[c] = df[c].fillna("Unknown")
    # Parse date-like columns when present
    for c in [col for col in df.columns if "date" in col.lower()]:
        df[c] = pd.to_datetime(df[c], errors="coerce")
    return df

raw = pd.read_csv(DATA_PATH, low_memory=False)
df = basic_clean(raw)
print("Rows:", len(df), "| Columns:", len(df.columns))
df.head()
```

```
Rows: 36403 | Columns: 18
```

Out[3]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	l
0	2539	Superfast Wi-Fi. Clean & quiet home by the park	2787	John	Brooklyn	Kensington	40.64529	-
1	2595	Skylit Studio Oasis | Midtown Manhattan	2845	Jennifer	Manhattan	Midtown	40.75356	-
2	6848	Only 2 stops to Manhattan studio	15991	Allen	Brooklyn	Williamsburg	40.70935	-
3	6872	Uptown Sanctuary w/ Private Bath (Month to Month)	16104	Kahshanna	Manhattan	East Harlem	40.80107	-
4	6990	UES Beautiful Blue Room	16800	Cynthia	Manhattan	East Harlem	40.78778	-

Save cleaned CSV and summary statistics

In [4]:
```
cleaned_csv_path = os.path.join(OUT_DIR, "cleaned_listings.csv")
df.to_csv(cleaned_csv_path, index=False)
print("Saved:", cleaned_csv_path)

summary = df.select_dtypes(include=[np.number]).describe()
summary_path = os.path.join(OUT_DIR, "summary_stats.csv")
summary.to_csv(summary_path)
print("Saved:", summary_path)

summary.head()
```

```
Saved: outputs/cleaned_listings.csv
Saved: outputs/summary_stats.csv
```

Out[4]:

	id	host_id	latitude	longitude	price	minimum_ni
count	3.640300e+04	3.640300e+04	36403.000000	36403.000000	36403.000000	36403.00
mean	4.525526e+17	1.737617e+08	40.728443	-73.947333	324.117792	28.61
std	5.320711e+17	1.922877e+08	0.056336	0.055033	2431.257283	29.28
min	2.539000e+03	1.678000e+03	40.500366	-74.251907	3.000000	1.00
25%	2.147279e+07	1.777536e+07	40.688320	-73.983517	130.000000	30.00

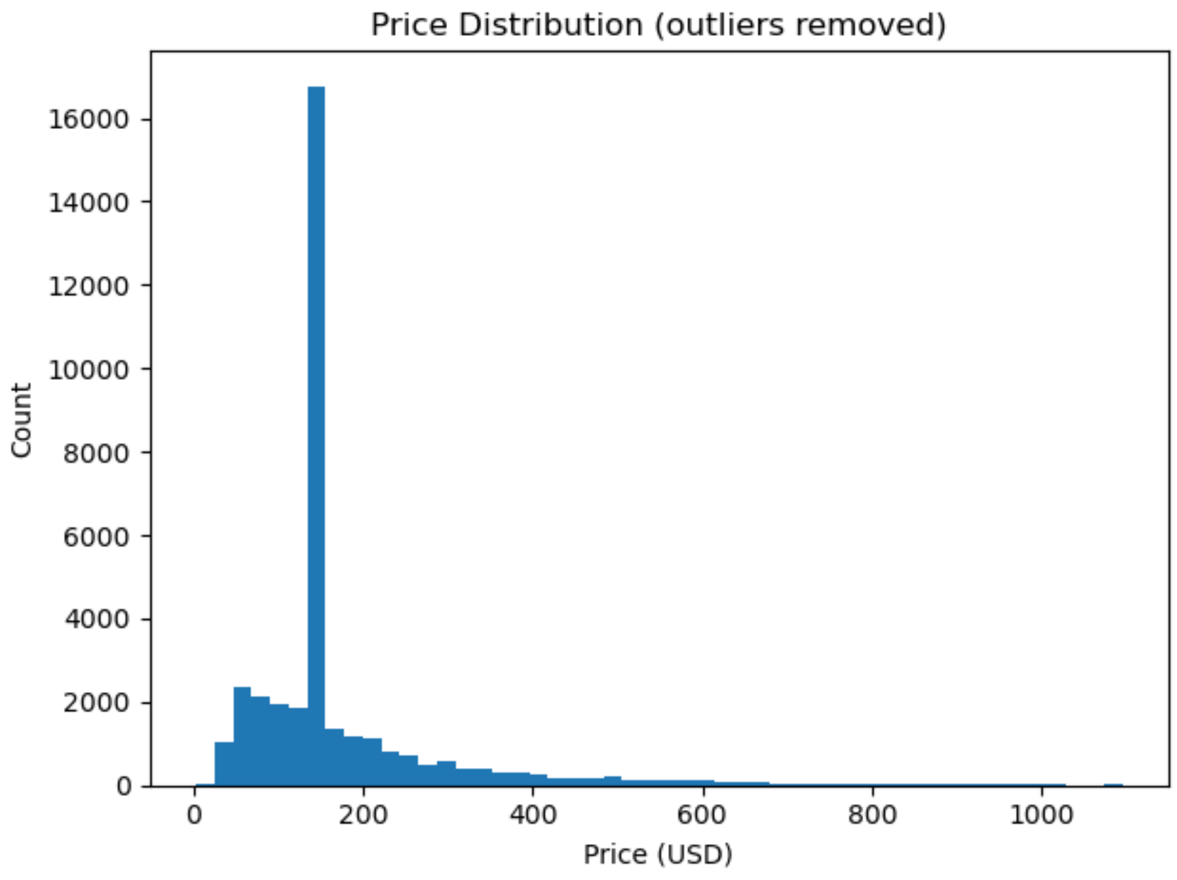
KPIs

In [6]:
```python
def safe_col(df, candidates, default=None):
    for c in candidates:
        if c in df.columns:
            return c
    return default

price_col = safe_col(df, ["price"])
neigh_col = safe_col(df, ["neighbourhood_cleansed", "neighbourhood"
room_col  = safe_col(df, ["room_type"])
reviews_col = safe_col(df, ["number_of_reviews"])
avail_col = safe_col(df, ["availability_365"])

kpis = {
    "Total Listings": len(df),
    "Average Price": float(df[price_col].mean()) if price_col else
    "Average Reviews": float(df[reviews_col].mean()) if reviews_col
    "Average Availability 365": float(df[avail_col].mean()) if avai
}
kpis
```

Out[6]:
```
{'Total Listings': 36403,
 'Average Price': 324.11779248962995,
 'Average Reviews': 26.85105623162926,
 'Average Availability 365': 161.65516578303985}
```

Price distribution

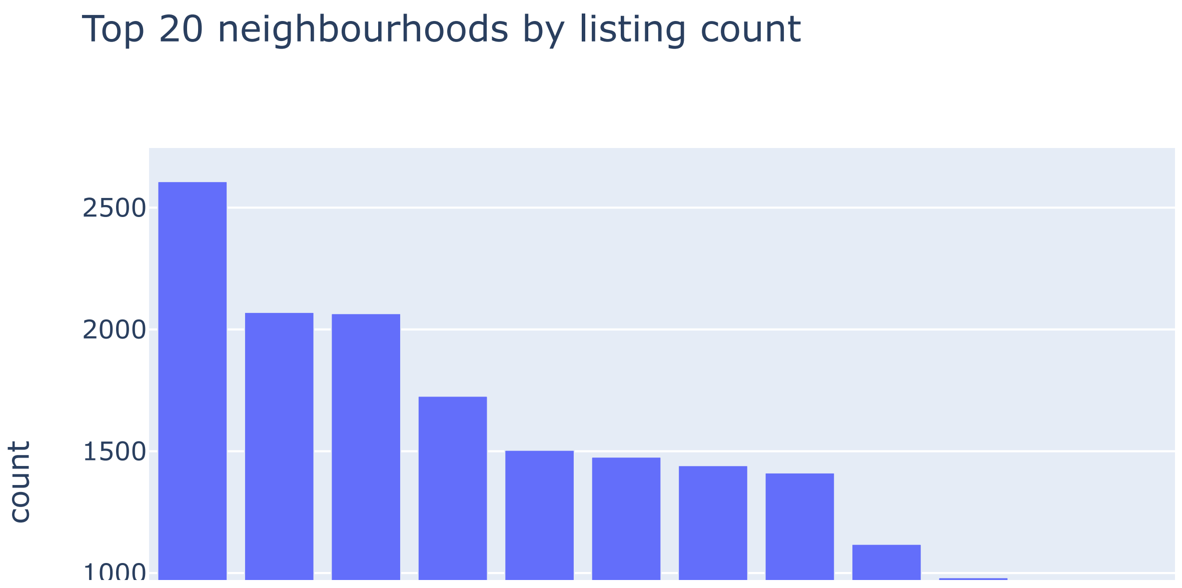
In [8]:
```python
if price_col:
    # Remove extreme outliers (e.g., top 1%)
    price_series = df[price_col].dropna()
    cutoff = price_series.quantile(0.99)
    filtered_prices = price_series[price_series <= cutoff]

    plt.figure()
    filtered_prices.plot(kind="hist", bins=50)
    plt.xlabel("Price (USD)")
    plt.ylabel("Count")
    plt.title("Price Distribution (outliers removed)")
    plt.tight_layout()
    out = os.path.join(OUT_DIR, "price_hist_filtered.png")
    plt.savefig(out, dpi=150)
    plt.show()
    print("Saved:", out)
else:
    print("No price column found.")
```



```
Saved: outputs/price_hist_filtered.png
```

Listings by neighbourhood

In [9]:
```
if neigh_col:
    topN = 20
    vc = df[neigh_col].value_counts().head(topN).reset_index()
    vc.columns = [neigh_col, "count"]
    fig = px.bar(vc, x=neigh_col, y="count", title=f"Top {topN} nei
    fig.show()
    out_html = os.path.join(OUT_DIR, "listings_by_neighbourhood.htm
    fig.write_html(out_html, include_plotlyjs="cdn")
    print("Saved:", out_html)
else:
    print("No neighbourhood column found.")
```

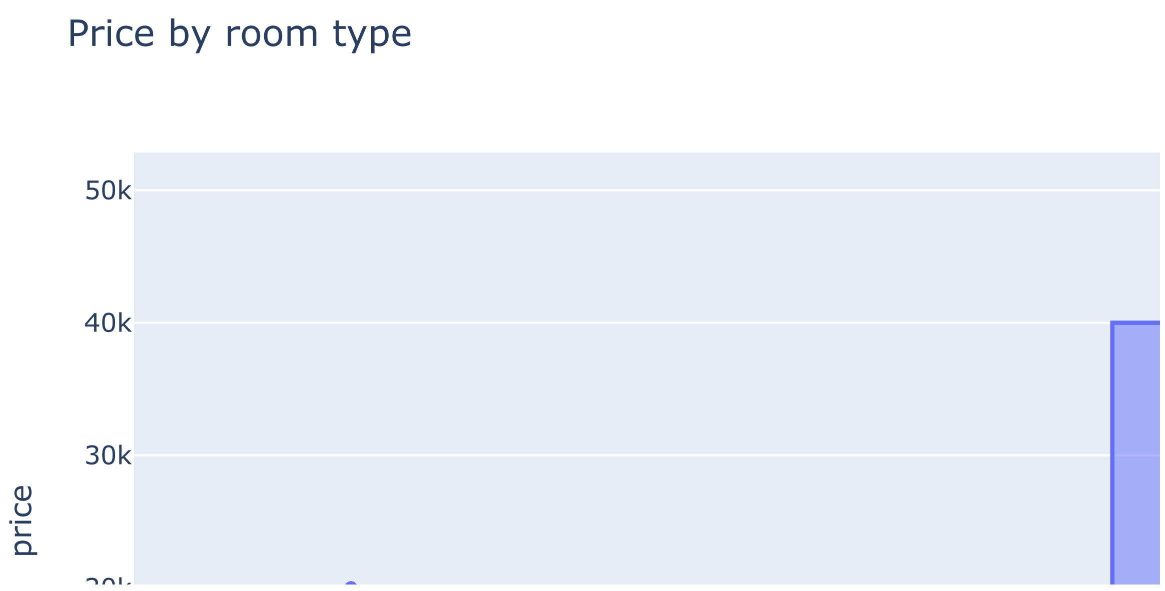


```
Saved: outputs/listings_by_neighbourhood.html
```

Price by room type

In [10]:

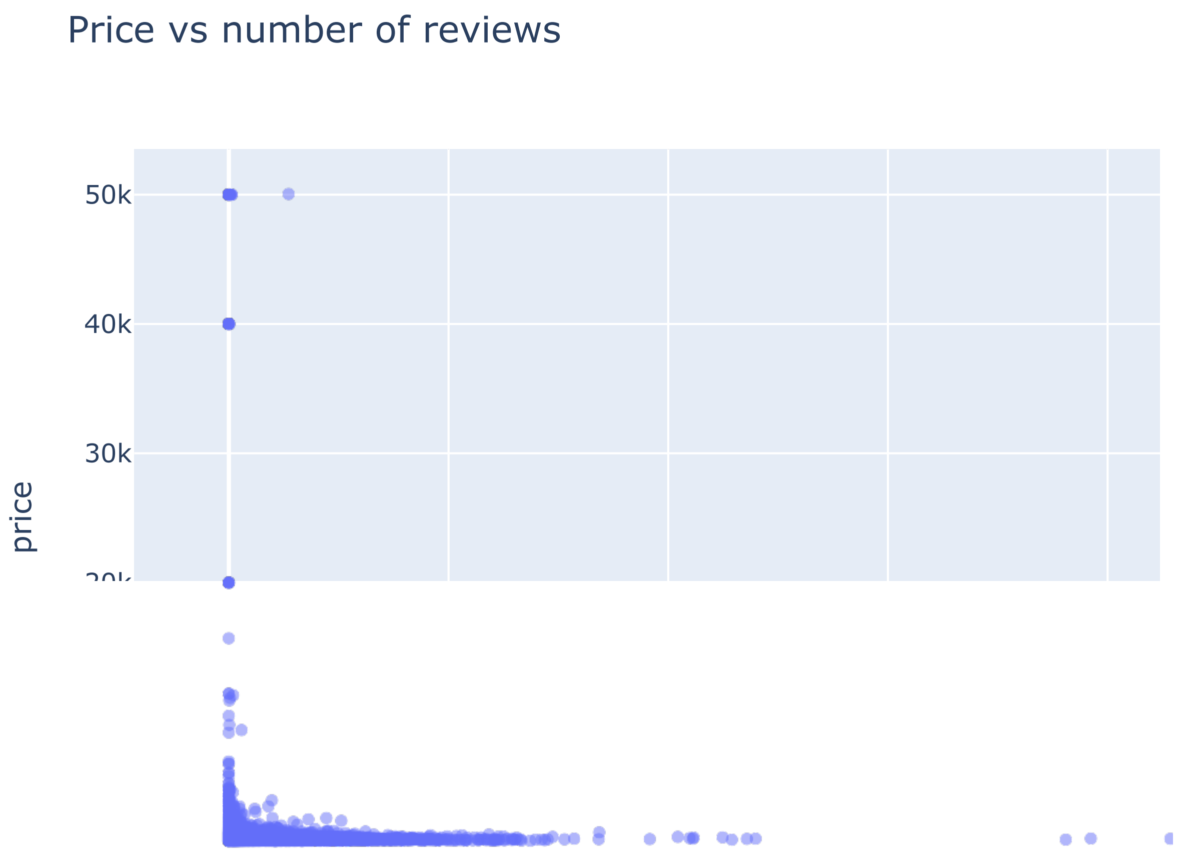
```
if price_col and room_col:
    fig = px.box(df[[room_col, price_col]].dropna(), x=room_col, y=
    fig.show()
    out_html = os.path.join(OUT_DIR, "price_by_room_type.html")
    fig.write_html(out_html, include_plotlyjs="cdn")
    print("Saved:", out_html)
else:
    print("Room type or price column missing.")
```



```
Saved: outputs/price_by_room_type.html
```

Price vs number of reviews

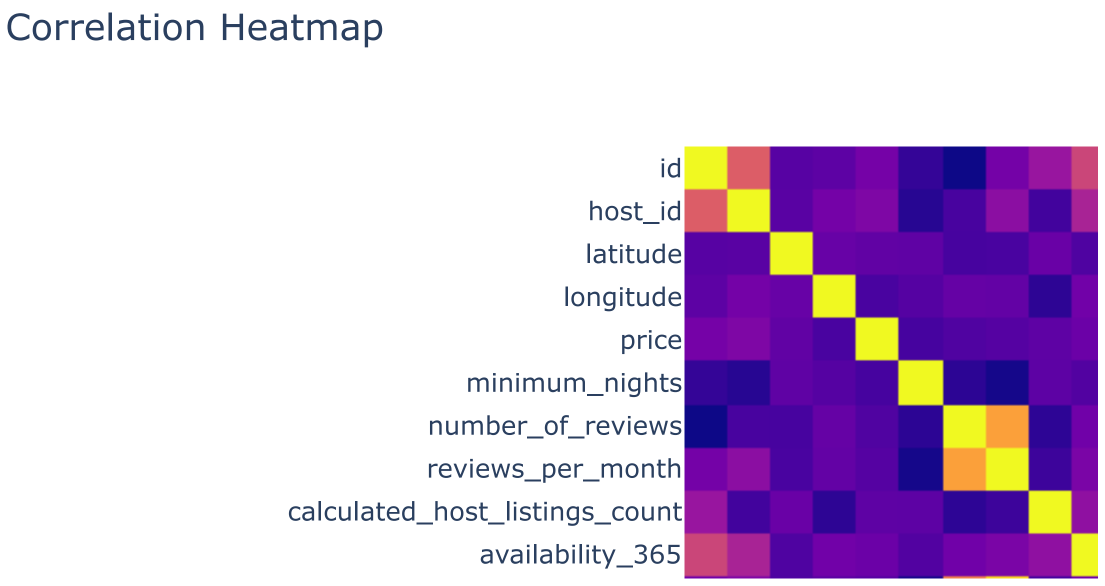
In [12]:
```python
if price_col and reviews_col:
    fig = px.scatter(
        df,
        x=reviews_col,
        y=price_col,
        opacity=0.5,
        title="Price vs number of reviews"
    )
    fig.show()
    out_html = os.path.join(OUT_DIR, "price_vs_reviews.html")
    fig.write_html(out_html, include_plotlyjs="cdn")
    print("Saved:", out_html)
```



```
Saved: outputs/price_vs_reviews.html
```

Correlation heatmap

In [13]:
```python
num_df = df.select_dtypes(include=[np.number])
if not num_df.empty:
    corr = num_df.corr(numeric_only=True)
    fig = px.imshow(corr, title="Correlation Heatmap")
    fig.show()
    out_html = os.path.join(OUT_DIR, "correlation_heatmap.html")
    fig.write_html(out_html, include_plotlyjs="cdn")
    print("Saved:", out_html)
else:
    print("No numeric columns for correlation.")
```



```
Saved: outputs/correlation_heatmap.html
```

Linear regression for price

In [16]:
```
if price_col:
    feature_candidates = [c for c in ["minimum_nights", "maximum_ni
    if feature_candidates:
        X = df[feature_candidates].fillna(0.0)
        y = df[price_col].fillna(df[price_col].median())
        X_train, X_test, y_train, y_test = train_test_split(X, y, t
        model = LinearRegression().fit(X_train, y_train)
        r2 = model.score(X_test, y_test)
        print("R^2 on hold-out:", round(r2, 3))
        coef = pd.DataFrame({"feature": feature_candidates, "coef":
        display(coef)
    else:
        print("Not enough numeric features for regression.")
else:
    print("Price column is required for regression.")
```

```
R^2 on hold-out: 0.008
```

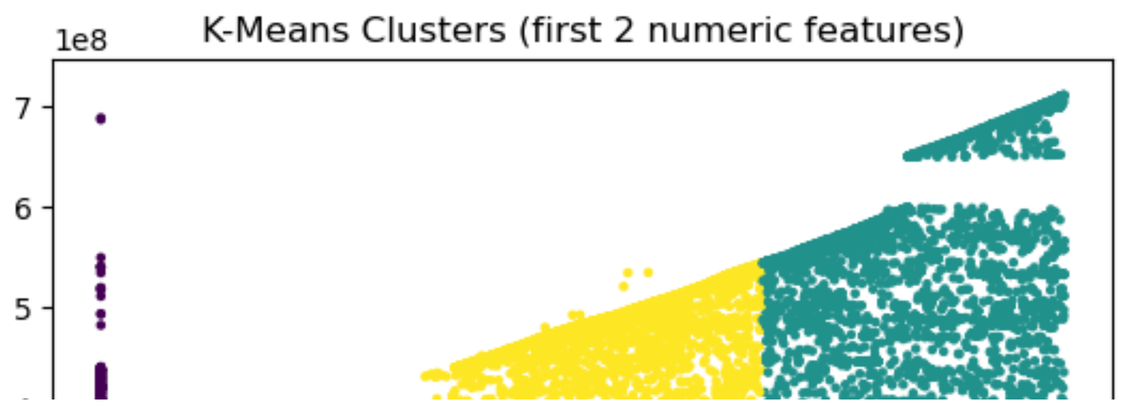
	feature	coef
1	availability_365	0.939912
3	calculated_host_listings_count	0.086410
2	number_of_reviews	-1.123670
0	minimum_nights	-4.246441

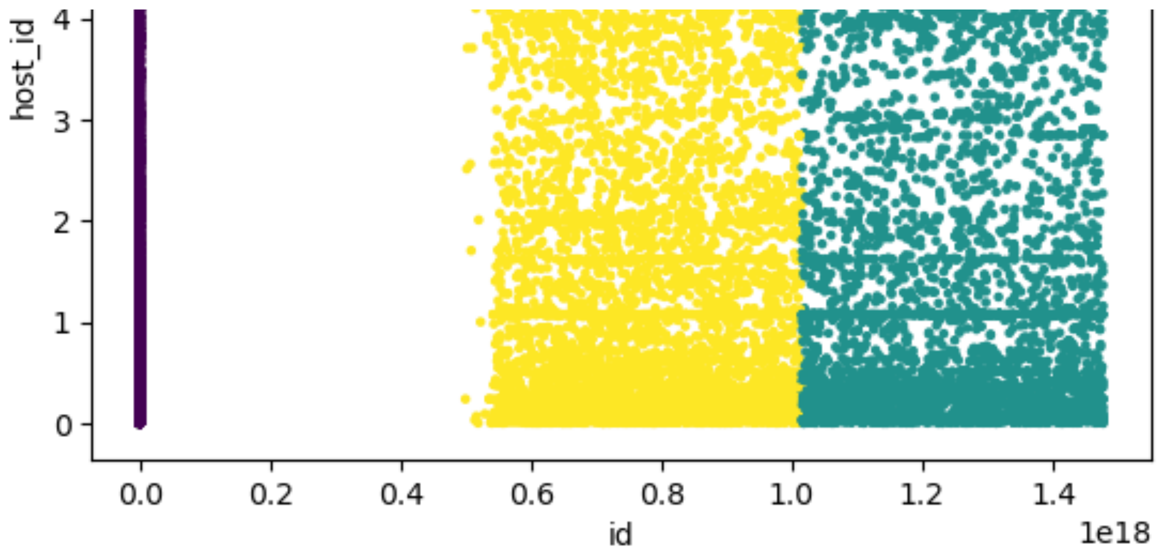
Simple K-Means clustering

In [19]:
```
num_df = df.select_dtypes(include=[np.number])
if not num_df.empty:
    k = 3
    km = KMeans(n_clusters=k, n_init="auto", random_state=42)
    fit_df = num_df.fillna(num_df.median(numeric_only=True))
    labels = km.fit_predict(fit_df)
    df_clu = df.copy()
    df_clu["cluster"] = labels
    print("Cluster counts:")
    print(df_clu["cluster"].value_counts())
    display(df_clu.head(10))
else:
    print("No numeric columns available for clustering.")

import matplotlib.pyplot as plt

plt.scatter(fit_df.iloc[:, 0], fit_df.iloc[:, 1], c=labels, cmap="v
plt.title("K-Means Clusters (first 2 numeric features)")
plt.xlabel(fit_df.columns[0])
plt.ylabel(fit_df.columns[1])
plt.show()
```

```
Cluster counts:
cluster
0    20100
2     8325
1     7978
Name: count, dtype: int64
```

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	l
0	2539	Superfast Wi-Fi. Clean & quiet home by the park	2787	John	Brooklyn	Kensington	40.
1	2595	Skylit Studio Oasis | Midtown Manhattan	2845	Jennifer	Manhattan	Midtown	40.
2	6848	Only 2 stops to Manhattan studio	15991	Allen	Brooklyn	Williamsburg	40.
3	6872	Uptown Sanctuary w/ Private Bath (Month to Month)	16104	Kahshanna	Manhattan	East Harlem	40.
4	6990	UES Beautiful Blue Room	16800	Cynthia	Manhattan	East Harlem	40.
5	7064	Amazing location! Wburg. Large, bright & tranquil	17297	Joelle	Brooklyn	Williamsburg	40.
6	7097	Perfect for Your Parents, With Garden & Patio	17571	Jane	Brooklyn	Fort Greene	40.
7	7801	Sunny Williamsburg Loft with Sauna	21207	Chaya	Brooklyn	Williamsburg	40.
8	8490	Maison des Sirenes1,bohemian, luminous apartment	25183	Nathalie	Brooklyn	Bedford-Stuyvesant	40.
9	9357	Midtown Pied-a-terre	30193	Tommi Laurelle	Manhattan	Hell's Kitchen	40.





Export example filtered CSV

In [20]:
```
room_col = room_col or "room_type" if "room_type" in df.columns els
if room_col and price_col:
    filtered = df[(df[room_col] == "Entire home/apt") & (df[price_c
    filtered_path = os.path.join(OUT_DIR, "filtered_listings.csv")
    filtered.to_csv(filtered_path, index=False)
    print("Saved:", filtered_path, "| Rows:", len(filtered))
else:
    print("Cannot create filtered CSV because room_type or price is
```

```
Saved: outputs/filtered_listings.csv | Rows: 16288
```

How to run the Streamlit dashboard

The Streamlit dashboard is provided in `app.py`. It can auto-download the same NYC dataset if `listings.csv` is missing.

Run these commands in terminal from the project folder:

```
pip install -r requirements.txt
python fetch_nyc_data.py
streamlit run app.py
```

Then open the local URL that Streamlit prints (usually http://localhost:8501 (http://localhost:8501)). Use the sidebar to filter and the buttons to export `cleaned_listings.csv` and `filtered_listings.csv`.

In [ ]:
```
```