CST383 Final Project — Predicting Satellite Mission Type from Orbital & Physical Features

Team: Jess Hammond, Gary Kuepper, Keshab Neupane **Course:** CST 383 — Data Science
Video (≤5 min): *link*

Introduction

This project explores whether a satellite's **mission type** (e.g., *Communications*, *Navigation*, *Science*, *Technology Demonstration*, *Surveillance*) can be predicted using only its **orbital and physical parameters**.

Motivation

1. **Rapid Cataloging:** Public and private organizations manage thousands of satellites. A predictive model could accelerate cataloging and assist in identifying unknown satellites.
2. **Scientific Insight:** Understanding relationships between orbit design and mission type reveals trends in satellite engineering (e.g., *Geostationary* orbits for communications vs. *Sun-synchronous* for science missions).

Research Question

Can we accurately predict a satellite's **mission category** using publicly available **orbital and physical features** from the UCS Satellite Database?

In [59]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.pyplot import title
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

Read the Data

Data Characteristics:

- ~7,560 merged entries (UCS + SatCat)
- Key features: `OrbitClass`, `OrbitType`, `Apogee`, `Perigee`, `Eccentricity`, `Inclination`, `Period`, `LaunchMass`, `LaunchYear`
- **Target:** `Purpose` (grouped into 6 super-classes) | Category | Count | |-----------|------:| | Communications | 5503 | | Science | 1093 | | Tech Demo | 453 | | Navigation | 345 | | Surveillance | 156 | | Other | 10 |

In [60]:
```
# UCS Satellite Database
# url: https://www.ucs.org/resources/satellite-database
url = "UCS-Satellite-Database-Officialname_5-1-2023.xlsx"
df_ucs = pd.read_excel(url, sheet_name="Sheet1")
print(df_ucs.info())

# Satcat Data
# url: https://www.celestrak.org/satcat/satcat-format.php

file = "satcat.csv"
df_satcat = pd.read_csv(file)
print(df_satcat.info())
df = pd.merge(df_ucs, df_satcat, left_on='NORAD Number', right_on='NORAD_CAT_ID')

# df.to_csv("merged_data.csv", index=False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7560 entries, 0 to 7559
Data columns (total 67 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   Current Official Name of Satellite  7560 non-null   object 
 1   Country/Org of UN Registry         7559 non-null   object 
 2   Country of Operator/Owner          7560 non-null   object 
 3   Operator/Owner                     7560 non-null   object 
 4   Users                              7560 non-null   object 
 5   Purpose                            7560 non-null   object 
 6   Detailed Purpose                   1254 non-null   object 
 7   Class of Orbit                     7560 non-null   object 
 8   Type of Orbit                      6909 non-null   object 
 9   Longitude of GEO (degrees)         7557 non-null   float64
 10  Perigee (km)                       7553 non-null   float64
 11  Apogee (km)                        7553 non-null   float64
 12  Eccentricity                       7549 non-null   float64
 13  Inclination (degrees)              7556 non-null   float64
 14  Period (minutes)                   7504 non-null   float64
 15  Launch Mass (kg.)                  7315 non-null   float64
 16  Dry Mass (kg.)                     767 non-null    object 
 17  Power (watts)                      579 non-null    object 
 18  Date of Launch                     7559 non-null   object 
 19  Expected Lifetime (yrs.)           5450 non-null   float64
 20  Contractor                         7560 non-null   object 
 21  Country of Contractor              7560 non-null   object 
 22  Launch Site                        7560 non-null   object 
 23  Launch Vehicle                     7560 non-null   object 
 24  COSPAR Number                      7560 non-null   object 
 25  NORAD Number                       7560 non-null   int64  
 26  Comments                           2085 non-null   object 
 27  Unnamed: 27                        5 non-null      object 
 28  Source Used for Orbital Data       6636 non-null   object 
 29  Source                             3286 non-null   object 
 30  Source.1                           725 non-null    object 
 31  Source.2                           1832 non-null   object 
 32  Source.3                           1126 non-null   object 
 33  Source.4                           729 non-null    object 
 34  Source.5                           553 non-null    object 
 35  Source.6                           504 non-null    object 
 36  Unnamed: 36                        484 non-null    object 
 37  Unnamed: 37                        484 non-null    object 
 38  Unnamed: 38                        484 non-null    object 
 39  Unnamed: 39                        484 non-null    object 
 40  Unnamed: 40                        484 non-null    object 
 41  Unnamed: 41                        484 non-null    object 
 42  Unnamed: 42                        484 non-null    object 
 43  Unnamed: 43                        484 non-null    object 
 44  Unnamed: 44                        484 non-null    object 
 45  Unnamed: 45                        484 non-null    object 
 46  Unnamed: 46                        484 non-null    object 
 47  Unnamed: 47                        484 non-null    object 
 48  Unnamed: 48                        484 non-null    object 
 49  Unnamed: 49                        484 non-null    object 
 50  Unnamed: 50                        484 non-null    object 
```

```
 51  Unnamed: 51                          484 non-null    object
 52  Unnamed: 52                          485 non-null    object
 53  Unnamed: 53                          485 non-null    object
 54  Unnamed: 54                          485 non-null    object
 55  Unnamed: 55                          485 non-null    object
 56  Unnamed: 56                          485 non-null    object
 57  Unnamed: 57                          485 non-null    object
 58  Unnamed: 58                          485 non-null    object
 59  Unnamed: 59                          488 non-null    object
 60  Unnamed: 60                          488 non-null    object
 61  Unnamed: 61                          487 non-null    object
 62  Unnamed: 62                          485 non-null    object
 63  Unnamed: 63                          485 non-null    object
 64  Unnamed: 64                          485 non-null    object
 65  Unnamed: 65                          487 non-null    object
 66  Unnamed: 66                          485 non-null    object
dtypes: float64(8), int64(1), object(58)
memory usage: 3.9+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65935 entries, 0 to 65934
Data columns (total 17 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   OBJECT_NAME       65935 non-null  object
 1   OBJECT_ID         65935 non-null  object
 2   NORAD_CAT_ID      65935 non-null  int64
 3   OBJECT_TYPE       65935 non-null  object
 4   OPS_STATUS_CODE   49145 non-null  object
 5   OWNER             65935 non-null  object
 6   LAUNCH_DATE       65935 non-null  object
 7   LAUNCH_SITE       65935 non-null  object
 8   DECAY_DATE        34244 non-null  object
 9   PERIOD            65012 non-null  float64
 10  INCLINATION       65012 non-null  float64
 11  APOGEE            65012 non-null  float64
 12  PERIGEE           65012 non-null  float64
 13  RCS               32931 non-null  float64
 14  DATA_STATUS_CODE  1244 non-null   object
 15  ORBIT_CENTER      65935 non-null  object
 16  ORBIT_TYPE        65935 non-null  object
dtypes: float64(5), int64(1), object(11)
memory usage: 8.6+ MB
None
```

Initial Exploration/Data Cleaning

Data Cleaning & Feature Engineering

- Dropped redundant *Unnamed* columns; merged fragmented *Source* fields.
- Normalized categorical labels — consolidated **Civil, Government, and Military** user types into consistent categories, which noticeably improved model accuracy.
- Incorporated additional **Owner** and **Operator** fields; large clusters such as **SpaceX/**

Starlink became highly visible on launch-year timeline plots.

- Parsed and extracted `LaunchYear` from `LaunchDate` .
- Imputed missing numeric values with medians.
- Used `DetailedPurpose` text to refine class assignments — reclassified **military intelligence** and reconnaissance satellites from *Science* to *Surveillance* for more accurate labeling.
- Encoded categoricals with one-hot encoding.
- Created engineered features:
  - `LaunchYear` (numeric)
  - Period/Inclination "buckets" for visualization
  - `PurposeSuperAudit` : manually curated high-level mission categories.

This cleans the table: we rename long column names to shorter ones, drop the junk "Unnamed:" *columns, merge all the Source* columns into one text field (SourcesAll), and convert LaunchDate to a real date so we can also make LaunchYear.

In [61]:
```
# Clean data
rename_map = {
    'Current Official Name of Satellite': 'SatelliteName',
    'Country/Org of UN Registry': 'UNRegistry',
    'Country of Operator/Owner': 'Country',
    'Operator/Owner': 'Operator',
    'Users': 'Users',
    'Purpose': 'Purpose',
    'Detailed Purpose': 'DetailedPurpose',
    'Class of Orbit': 'OrbitClass',
    'Type of Orbit': 'OrbitType',
    'Longitude of GEO (degrees)': 'GEOLongitude',
    'Perigee (km)': 'Perigee',
    'Apogee (km)': 'Apogee',
    'Eccentricity': 'Eccentricity',
    'Inclination (degrees)': 'Inclination',
    'Period (minutes)': 'Period',
    'Launch Mass (kg.)': 'LaunchMass',
    'Dry Mass (kg.)': 'DryMass',
    'Power (watts)': 'Power',
    'Date of Launch': 'LaunchDate',
    'Expected Lifetime (yrs.)': 'LifetimeYrs',
    'Contractor': 'Contractor',
    'Country of Contractor': 'ContractorCountry',
    'Launch Site': 'LaunchSite',
    'Launch Vehicle': 'LaunchVehicle',
    'COSPAR Number': 'COSPAR',
    'NORAD Number': 'NORAD',
    'Comments': 'Comments',
    # Satcat.csv columns
    'OBJECT_NAME': 'SatelliteName',
    'OBJECT_ID': 'ObjectID',
    'NORAD_CAT_ID': 'NORAD',
    'OBJECT_TYPE': 'ObjectType',
```

```
    'OPS_STATUS_CODE': 'OPSStatusCode',
    'OWNER': 'Owner',
    'LAUNCH_DATE': 'LaunchDate_sc',
    'LAUNCH_SITE': 'LaunchSite_sc',
    'DECAY_DATE': 'DecayDate',
    'PERIOD': 'Period_sc',
    'INCLINATION': 'Inclination_sc',
    'APOGEE': 'Apogee_sc',
    'PERIGEE': 'Perigee_sc',
    'RCS': 'RCS',
    'DATA_STATUS_CODE': 'DataStatusCode',
    'ORBIT_CENTER': 'OrbitCenter',
    'ORBIT_TYPE': 'OrbitType'
}
df = df.rename(columns={k: v for k, v in rename_map.items() if k in df.columns})
df['OrbitClass'] = df['OrbitClass'].astype(str).str.upper()
# drop all Unnamed: columns
unnamed_cols = [c for c in df.columns if c.startswith("Unnamed:")]
df = df.drop(columns=unnamed_cols)

# combine Source columns into one text field
source_cols = [c for c in df.columns if c.startswith("Source")]
if len(source_cols) > 0:
    df['SourcesAll'] = df[source_cols] \
        .astype(str) \
        .replace({'nan': np.nan}) \
        .apply(lambda row: "; ".join([v for v in row if pd.notna(v)]), axis=1)
    df = df.drop(columns=source_cols)

if 'LaunchDate' in df.columns:
    df['LaunchDate'] = pd.to_datetime(df['LaunchDate'], errors='coerce')
    df['LaunchYear'] = df['LaunchDate'].dt.year

print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7560 entries, 0 to 7559
Data columns (total 46 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   SatelliteName      7560 non-null   object
 1   UNRegistry         7559 non-null   object
 2   Country            7560 non-null   object
 3   Operator           7560 non-null   object
 4   Users              7560 non-null   object
 5   Purpose            7560 non-null   object
 6   DetailedPurpose    1254 non-null   object
 7   OrbitClass         7560 non-null   object
 8   OrbitType          6909 non-null   object
 9   GEOLongitude       7557 non-null   float64
 10  Perigee            7553 non-null   float64
 11  Apogee             7553 non-null   float64
 12  Eccentricity       7549 non-null   float64
 13  Inclination        7556 non-null   float64
 14  Period             7504 non-null   float64
 15  LaunchMass         7315 non-null   float64
 16  DryMass            767 non-null    object
 17  Power              579 non-null    object
 18  LaunchDate         7557 non-null   datetime64[ns]
 19  LifetimeYrs        5450 non-null   float64
 20  Contractor         7560 non-null   object
 21  ContractorCountry  7560 non-null   object
 22  LaunchSite         7560 non-null   object
 23  LaunchVehicle      7560 non-null   object
 24  COSPAR             7560 non-null   object
 25  NORAD              7560 non-null   int64
 26  Comments           2085 non-null   object
 27  SatelliteName      7560 non-null   object
 28  ObjectID           7560 non-null   object
 29  NORAD              7560 non-null   int64
 30  ObjectType         7560 non-null   object
 31  OPSStatusCode      7536 non-null   object
 32  Owner              7560 non-null   object
 33  LaunchDate_sc      7560 non-null   object
 34  LaunchSite_sc      7560 non-null   object
 35  DecayDate          1850 non-null   object
 36  Period_sc          7538 non-null   float64
 37  Inclination_sc     7538 non-null   float64
 38  Apogee_sc          7538 non-null   float64
 39  Perigee_sc         7538 non-null   float64
 40  RCS                1053 non-null   float64
 41  DataStatusCode     113 non-null    object
 42  OrbitCenter        7560 non-null   object
 43  OrbitType          7560 non-null   object
 44  SourcesAll         7560 non-null   object
 45  LaunchYear         7557 non-null   float64
dtypes: datetime64[ns](1), float64(14), int64(2), object(29)
memory usage: 2.7+ MB
None
```

Cleaning User column

In [62]:
```
# Clean up users data as there are many duplicates
print(f"Old Users")
print(df['Users'].value_counts(dropna=False))
syn_map = {
    'Govt': 'Government',
    'Gov': 'Government',
    'Government': 'Government',
    'Military': 'Military',
    'Commercial': 'Commercial',
    'Civil': 'Civil',
}

# Define a canonical order so joined categories are consistent
order = ['Commercial', 'Government', 'Military', 'Civil']
order_rank = {k: i for i, k in enumerate(order)}


def normalize_users(val):
    if pd.isna(val):
        return pd.NA
    # Split, trim, and standardize case
    tokens = [t.strip() for t in str(val).split('/')]
    tokens = [t for t in tokens if t != '']  # drop empties

    # Title-case to normalize, then apply synonyms map
    tokens = [syn_map.get(t.title(), t.title()) for t in tokens]

    # Deduplicate while keeping only known categories (optional: keep unknowns too)
    uniq = sorted(set(tokens), key=lambda x: order_rank.get(x, 999))

    if len(uniq) == 0:
        return pd.NA
    if len(uniq) == 1:
        return uniq[0]
    return '/'.join(uniq)


# Apply normalization (create a new column or overwrite existing)
df['Users'] = df['Users'].apply(normalize_users)

# If you want to overwrite the original column:
# df['Users'] = df['Users'].apply(normalize_users)

# Inspect results
print("\n New Clean Users")
print(df['Users'].value_counts(dropna=False))
```

```
Old Users
Users
Commercial                           6080
Government                            558
Military                              457
Civil                                 160
Government/Commercial                  97
Military/Commercial                    82
Military/Government                    56
Government/Civil                       40
Military/Civil                          7
Government/Military                     4
Commercial/Civil                        4
Civil/Government                        4
Civil/Military                          3
Commercial/Military                     2
Civil/Commercial                        1
Government/Commercial/Military          1
Commercial/Government                   1
Commercial                              1
Government                              1
Military                                1
Name: count, dtype: int64

 New Clean Users
Users
Commercial                           6081
Government                            559
Military                              458
Civil                                 160
Commercial/Government                  98
Commercial/Military                    84
Government/Military                    60
Government/Civil                       44
Military/Civil                         10
Commercial/Civil                        5
Commercial/Government/Military          1
Name: count, dtype: int64
```

Change OpstatusCode to verbose string

In [63]:
```python
# Example mapping (adjust if your codes or wording differ)
status_map = {
    '+': 'Op',
    '-': 'Nonop',
    'P': 'POp',
    'B': 'Bkp/Stb',
    'S': 'Spare',
    'X': 'ExtMis',
    'D': 'Decayed',
    '?': 'Unknown'
    # Add any other codes you have if needed
}
df['Status'] = df['OPSStatusCode'].map(status_map)
```

Clean up of Purpose Column

This part cleans the Purpose text (drops blanks, trims spaces, fixes casing), then groups many messy purpose labels into a few simple buckets: Communications, Navigation, Technology (incl. educational/platform), Science (Earth), Science (Space), and Science (Earth+Space). We apply the mapper to create PurposeSuperAudit, print the new bucket counts, and list anything that still fell into Other so we can fix it.

In [64]:
```
target_col = 'Purpose'

num_cols = [c for c in ['Perigee', 'Apogee', 'Eccentricity', 'Inclination', 'Launch
cat_cols = [c for c in ['OrbitClass', 'Users',  'Operator', 'Owner', 'Status'] if
            c in df.columns]

# Drop rows missing Purpose
dfm = df.dropna(subset=[target_col]).copy()
# Remove leading/trailing spaces and double spaces
dfm[target_col] = (
    dfm[target_col]
    .astype(str)
    .str.strip()
    .str.replace(r"\s+", " ", regex=True)
)

# Standardize capitalization
dfm[target_col] = dfm[target_col].str.title()

# Old buckets
num_of_old_buckets = dfm[target_col].value_counts()
print(f"\nOld Buckets: {len(num_of_old_buckets)}")


# print(dfm['DetailedPurpose'].value_counts().head(50))


# Combine purposes and remove redundant labels (Eg. Space Science & Space Observati
def map_purpose(p):
    s = str(p).lower().strip()
    s = " ".join(s.split())

    # Buckets
    if any(k in s for k in ["surveillance"]):
        return "Surveillance"

    if any(k in s for k in ["navigation", "positioning"]):
        return "Navigation"

    if any(k in s for k in ["mission extension", "technology", "educational", "educ
        return "Tech Demo"

    if any(k in s for k in ["science", "space", "earth", "meteorolog"]):
        return "Science"

    if any(k in s for k in ["communications", "communication"]):
```

```
            return "Communications"

    return "Other"


dfm["PurposeSuperAudit"] = dfm["Purpose"].apply(map_purpose)

# New buckets
print("\nNew Buckets")
print(dfm["PurposeSuperAudit"].value_counts())
```

```
Old Buckets: 30

New Buckets
PurposeSuperAudit
Communications    5519
Science           1390
Tech Demo          455
Navigation         166
Surveillance        20
Other               10
Name: count, dtype: int64
```

Further clean up of Military and Military satellites used for surveillance

In [65]:

```
# Keywords to detect in DetailedPurpose that should override PurposeSuperAudit
print("\nOld Buckets")
print(dfm["PurposeSuperAudit"].value_counts())

override_surveil = [
    "intelligence", "surveillance", "military", "reconnaissance",
    "comint", "signals", "spectrum monitoring", "military intelligence"
]

# AIS is maritime navigation system
override_nav = ["ais", "automatic identification system", "identification "]

def derive_from_detailed(p):
    s = str(p).lower().strip()
    s = " ".join(s.split())

    if any(k in s for k in override_surveil):
        # You can choose the target bucket here; using "Surveillance" as example
        return "Surveillance"

    if any(k in s for k in override_nav):
        # You can choose the target bucket here; using "Surveillance" as example
        return "Navigation"

    return None


# Compute overrides from DetailedPurpose
override_fetched = dfm["DetailedPurpose"].apply(derive_from_detailed)

# Apply overrides to existing PurposeSuperAudit, keeping original when no override
```

```
dfm["PurposeSuperAudit"] = override_fetched.where(override_fetched.notna(), dfm["Pu

# Ensure there is a default for any remaining unknowns
dfm["PurposeSuperAudit"] = dfm["PurposeSuperAudit"].fillna("Other")

print("\nNew Buckets")
print(dfm["PurposeSuperAudit"].value_counts())
```

```
Old Buckets
PurposeSuperAudit
Communications    5519
Science           1390
Tech Demo          455
Navigation         166
Surveillance        20
Other               10
Name: count, dtype: int64

New Buckets
PurposeSuperAudit
Communications    5503
Science           1093
Tech Demo          453
Navigation         345
Surveillance       156
Other               10
Name: count, dtype: int64
```

Operational Status of Satellites grouped by Purpose

In [66]:

```
# dfm.to_csv("merged_clean_data.csv", index=False)
table = dfm.groupby('PurposeSuperAudit')['Status'].value_counts(dropna=False).unsta
table['Total'] = table.sum(axis=1)
table = table.sort_values(by='Total', ascending=False)
with pd.option_context('display.max_columns', None):
    print(table)
print("\n", table)
```

```
Status              Bkp/Stb  Decayed  ExtMis  Nonop    Op  POp  Spare  NaN  \
PurposeSuperAudit
Communications            4     1039       0     97  4120  240      0    3
Science                   4      432       9     40   595    4      0    9
Tech Demo                 0      225       0     18   204    0      0    6
Navigation                9      138       0     18   171    5      4    0
Surveillance              0       15       0      4   131    0      0    6
Other                     0        1       0      0     9    0      0    0

Status              Total
PurposeSuperAudit
Communications       5503
Science              1093
Tech Demo             453
Navigation            345
Surveillance          156
Other                  10
```


```
 Status              Bkp/Stb  Decayed  ExtMis  Nonop    Op  POp  Spare  NaN  \
PurposeSuperAudit
Communications            4     1039       0     97  4120  240      0    3
Science                   4      432       9     40   595    4      0    9
Tech Demo                 0      225       0     18   204    0      0    6
Navigation                9      138       0     18   171    5      4    0
Surveillance              0       15       0      4   131    0      0    6
Other                     0        1       0      0     9    0      0    0

Status              Total
PurposeSuperAudit
Communications       5503
Science              1093
Tech Demo             453
Navigation            345
Surveillance          156
Other                  10
```

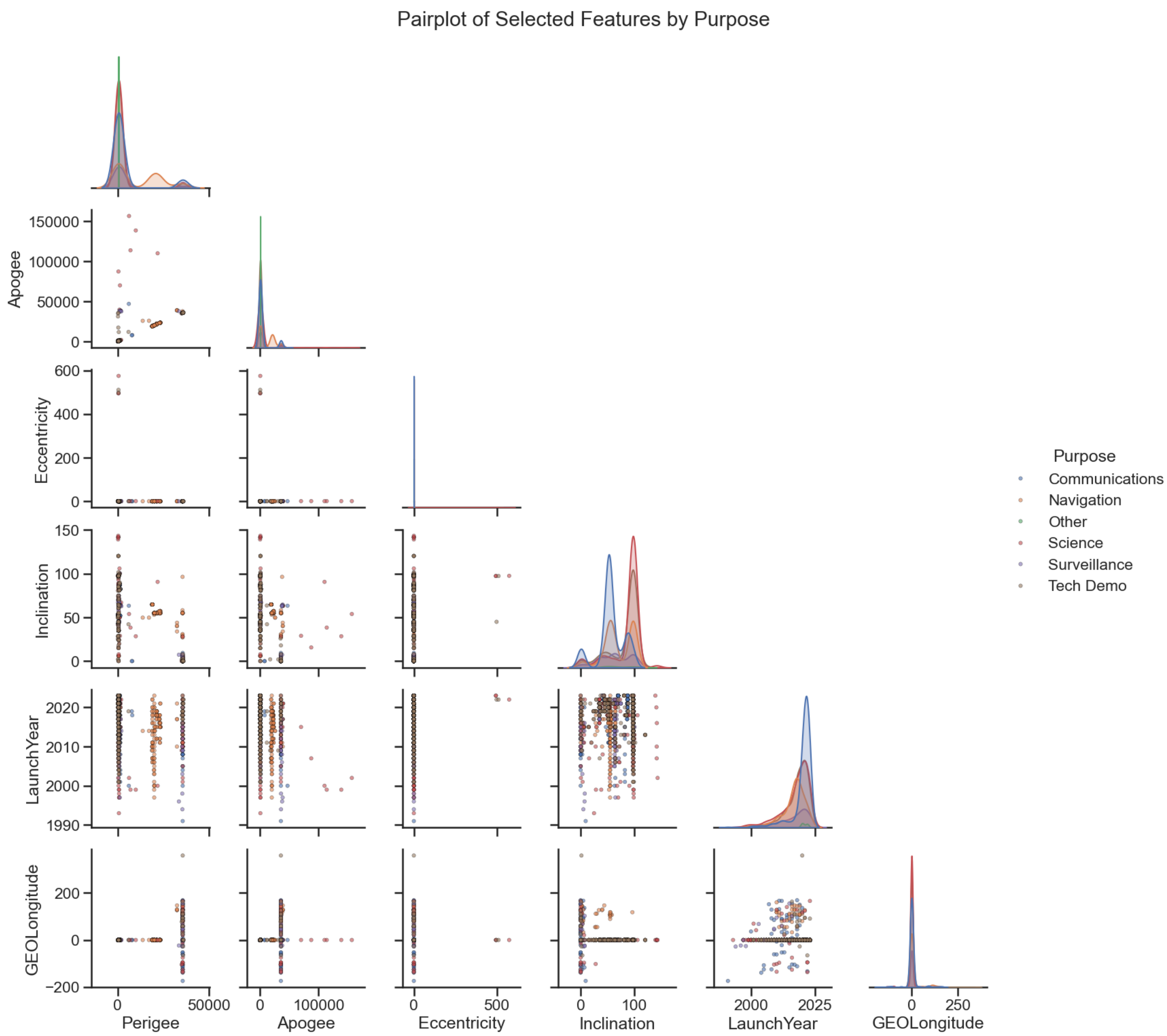

Exploration and Visualization

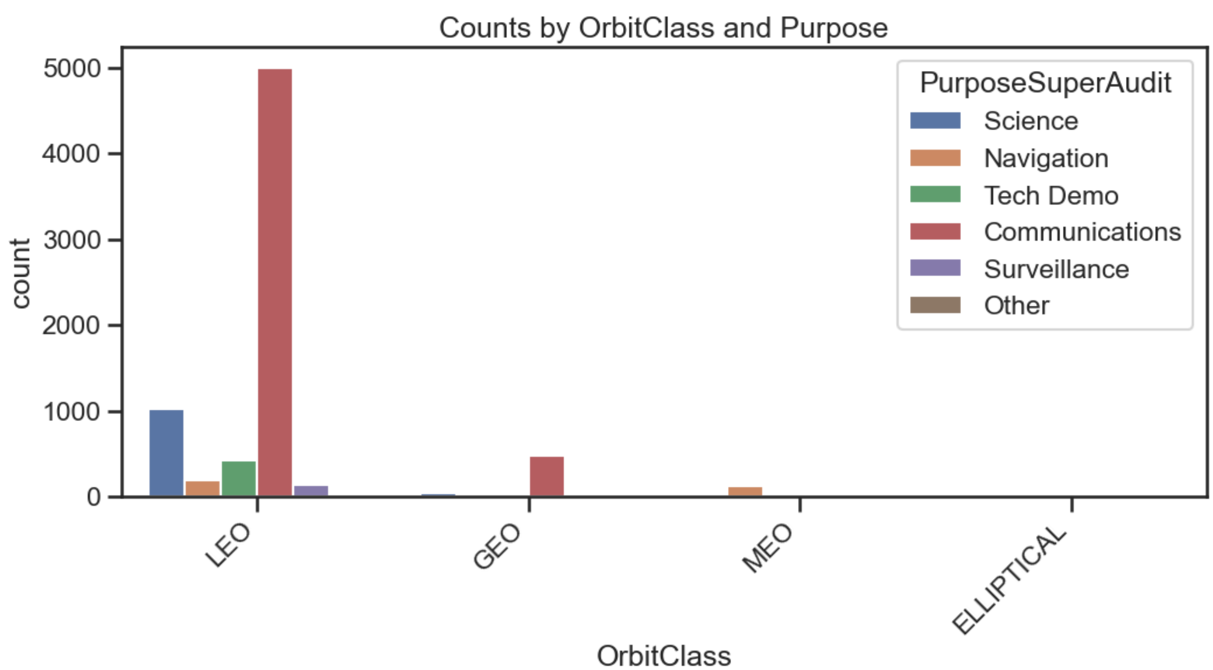
In [67]:

```
target_col = 'PurposeSuperAudit'  # or 'Purpose' if you prefer
max_points_per_class = 500

# Drop Unknown and ensure numerics present
df_plot = dfm.copy()
df_plot = df_plot[~df_plot[target_col].astype(str).str.lower().eq('unknown')]
df_plot = df_plot[num_cols + [target_col] + cat_cols].dropna(subset=num_cols)

# Downsample per class while preserving columns
df_plot_bal = (
    df_plot.groupby(target_col, group_keys=False)
    .sample(frac=1.0, random_state=42)  # shuffle within each class
    .groupby(target_col, group_keys=False)
    .head(max_points_per_class)  # cap per class
)

# Sanity checks
```

```python
print('Hue column present?', target_col in df_plot_bal.columns)
print('Class sizes used in plot:')
print(df_plot_bal[target_col].value_counts())

# Make the pairplot
# sns.set(style='ticks', context='talk')
g = sns.pairplot(
    df_plot_bal,
    vars=num_cols,
    hue=target_col if target_col in df_plot_bal.columns else None,
    corner=True,
    diag_kind='kde',
    plot_kws={'alpha': 0.6, 's': 15, 'edgecolor': 'black'}
)
g.fig.suptitle('Pairplot of Selected Features by Purpose', y=1.02)
g._legend.set_title("Purpose")


plt.show()

sns.set(style='ticks', context='talk')
n = len(cat_cols)

plot_df = dfm.copy()
plot_df[target_col] = plot_df[target_col].astype(str).str.strip()
plot_df = plot_df[~plot_df[target_col].str.lower().eq('unknown')]

# Optional: order categories by total count
order_oc = plot_df['OrbitClass'].value_counts().index

plt.figure(figsize=(12, 5))
sns.countplot(data=plot_df, x='OrbitClass', hue=target_col, order=order_oc)
plt.title('Counts by OrbitClass and Purpose')
plt.xticks(rotation=45, ha='right')

plt.show()
```

```
Hue column present? True
Class sizes used in plot:
PurposeSuperAudit
Communications    500
Science           500
Tech Demo         451
Navigation        345
Surveillance      156
Other              10
Name: count, dtype: int64
```





Training a Logistic Regression Model

Libraries: `pandas`, `NumPy`, `scikit-learn`, `matplotlib`, `seaborn`

Preprocessing Pipeline

1. **Numerical Features:** imputed with median → standardized (`StandardScaler`)
2. **Categorical Features:** one-hot encoded (`OneHotEncoder(handle_unknown='ignore')`)
3. **Split:** stratified 80 / 20 train–test split (preserving class balance)

Model Choice

- **Logistic Regression (multinomial, solver='lbfgs', max_iter=600)**
  - Serves as an **interpretable baseline** to evaluate how orbital and physical variables drive classification.
  - Chosen for simplicity and transparency before exploring non-linear models (RandomForest, CatBoost).

In [68]:
```
target_col = "PurposeSuperAudit"

# Drop Unknown label
dfm_model = dfm[~dfm[target_col].isin(["Unknown"])].copy()

X = dfm_model[num_cols + cat_cols].copy()
y = dfm_model[target_col].copy()

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=

# Impute numerics with median
for c in num_cols:
    med = X_train[c].median()
    X_train[c] = X_train[c].fillna(med)
    X_test[c] = X_test[c].fillna(med)

X_train[cat_cols] = X_train[cat_cols].astype(str)
X_test[cat_cols] = X_test[cat_cols].astype(str)

# Trun orbitalclass into 0 or 1
ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
ohe.fit(X_train[cat_cols])
Z_train_cat = ohe.transform(X_train[cat_cols])
Z_test_cat = ohe.transform(X_test[cat_cols])

# Scale numerics
scaler = StandardScaler().fit(X_train[num_cols])
Z_train_num = scaler.transform(X_train[num_cols])
Z_test_num = scaler.transform(X_test[num_cols])

# Combine features
Z_train = np.hstack([Z_train_num, Z_train_cat])
Z_test = np.hstack([Z_test_num, Z_test_cat])

lr = LogisticRegression(max_iter=600, solver='lbfgs',
```

```
                        class_weight=None)  # Tried class_weight='balance' but gave
lr.fit(Z_train, y_train)
```

Out[68]:

▾ LogisticRegression

▸ Parameters

	penalty	'l2'
	dual	False
	tol	0.0001
	C	1.0
	fit_intercept	True
	intercept_scaling	1
	class_weight	None
	random_state	None
	solver	'lbfgs'
	max_iter	600
	multi_class	'deprecated'
	verbose	0
	warm_start	False
	n_jobs	None
	l1_ratio	None

Model Performance

In [69]:

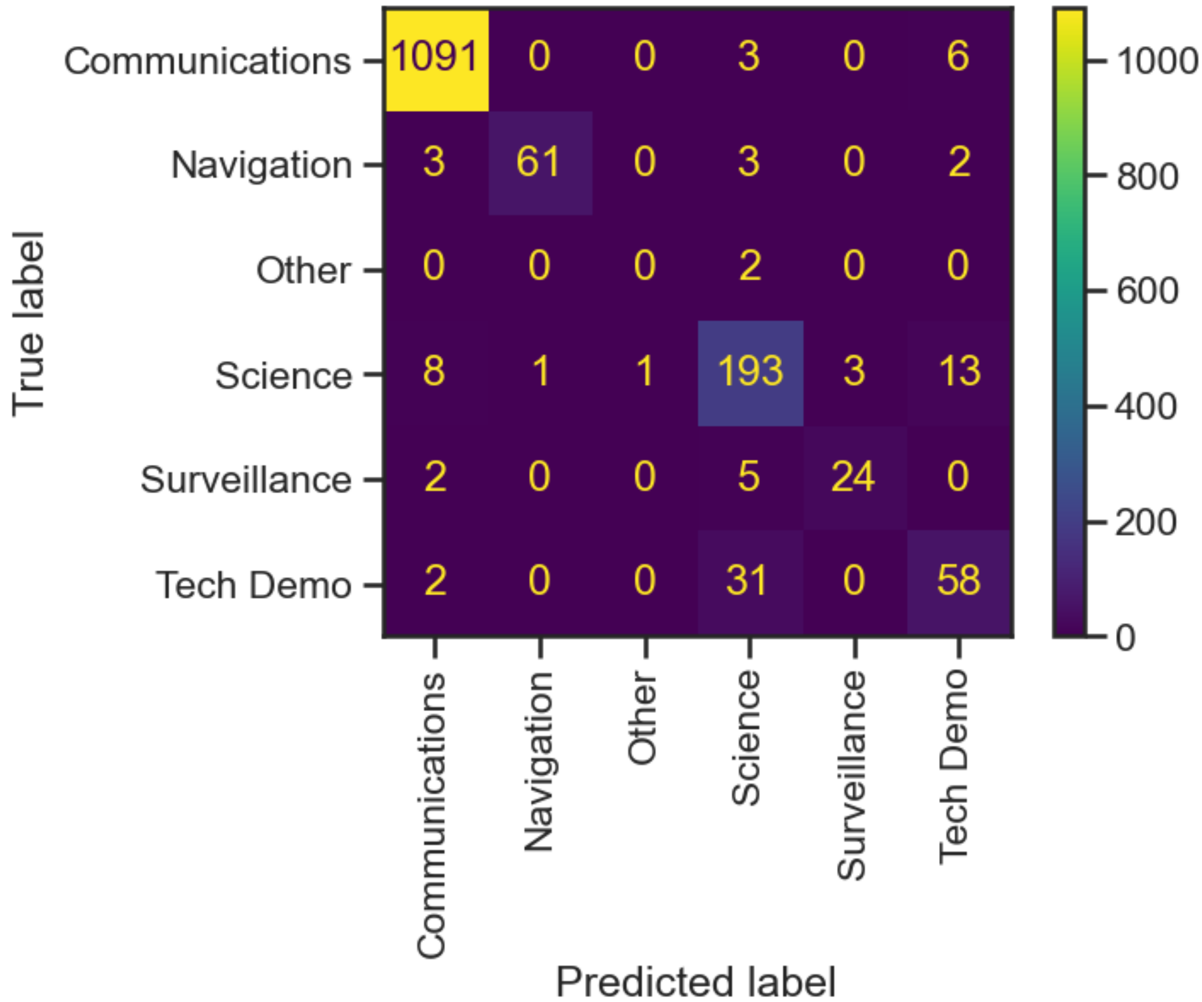
```
pred = lr.predict(Z_test)

print(f"\nAccuracy: {accuracy_score(y_test, pred):.3f}", )
print(classification_report(y_test, pred, zero_division=0))
```

```
Accuracy: 0.944
                precision    recall  f1-score   support

Communications       0.99      0.99      0.99      1100
    Navigation       0.98      0.88      0.93        69
         Other       0.00      0.00      0.00         2
       Science       0.81      0.88      0.85       219
  Surveillance       0.89      0.77      0.83        31
     Tech Demo       0.73      0.64      0.68        91

      accuracy                           0.94      1512
     macro avg       0.73      0.69      0.71      1512
  weighted avg       0.94      0.94      0.94      1512
```

In [70]:

```
# confusion matrix
labels = sorted(y_test.unique())
cm = confusion_matrix(y_test, pred, labels=labels)
print(labels)
print(cm)

labels = sorted(y_test.unique())
ConfusionMatrixDisplay.from_predictions(y_test, pred, labels=labels, xticks_rotatio
plt.show()
```

```
['Communications', 'Navigation', 'Other', 'Science', 'Surveillance', 'Tech Demo']
[[1091    0    0    3    0    6]
 [   3   61    0    3    0    2]
 [   0    0    0    2    0    0]
 [   8    1    1  193    3   13]
 [   2    0    0    5   24    0]
 [   2    0    0   31    0   58]]
```



Interpretation

- Model achieves **excellent performance** for dominant categories (*Communications*, *Navigation*).
- Misclassifications mostly between *Science* and *Tech Demo*, reflecting overlapping orbital patterns.
- “Other” category too small for meaningful learning.

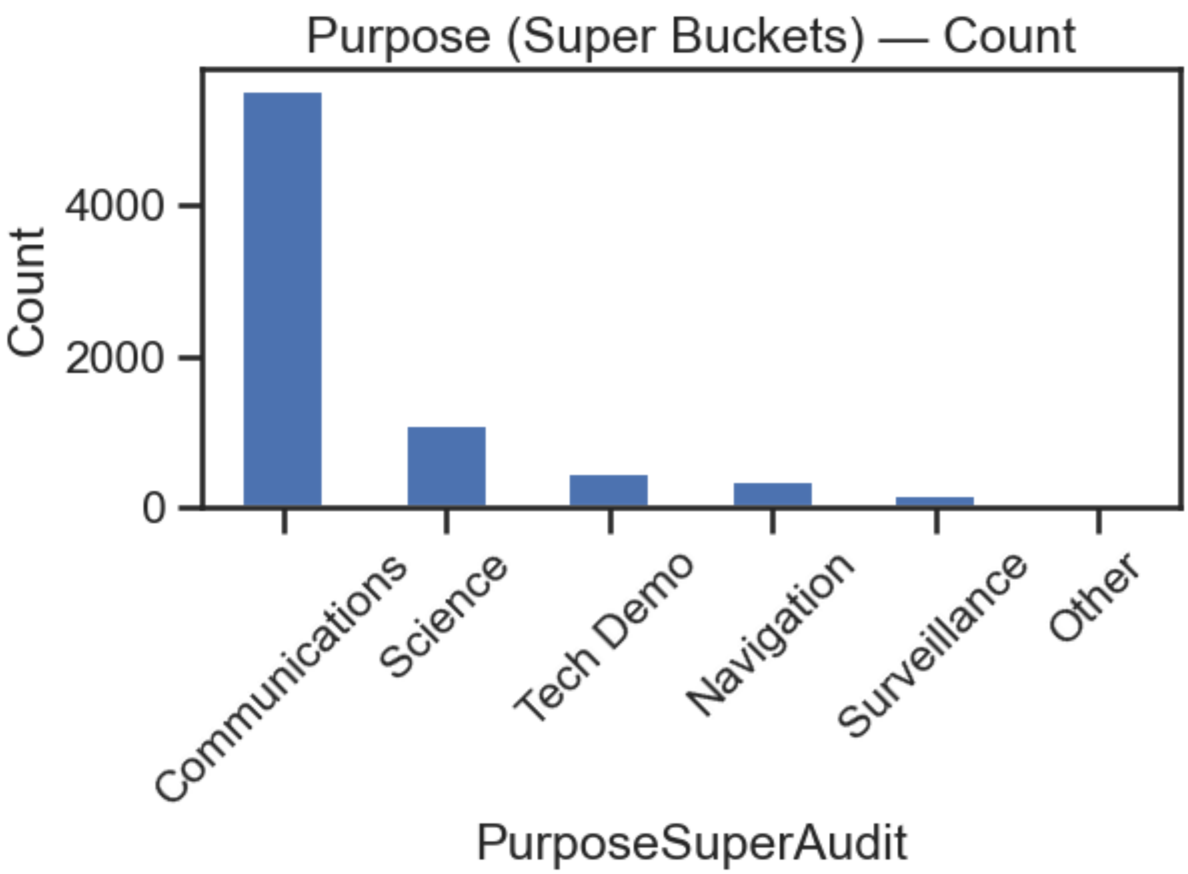
Visualization Highlights

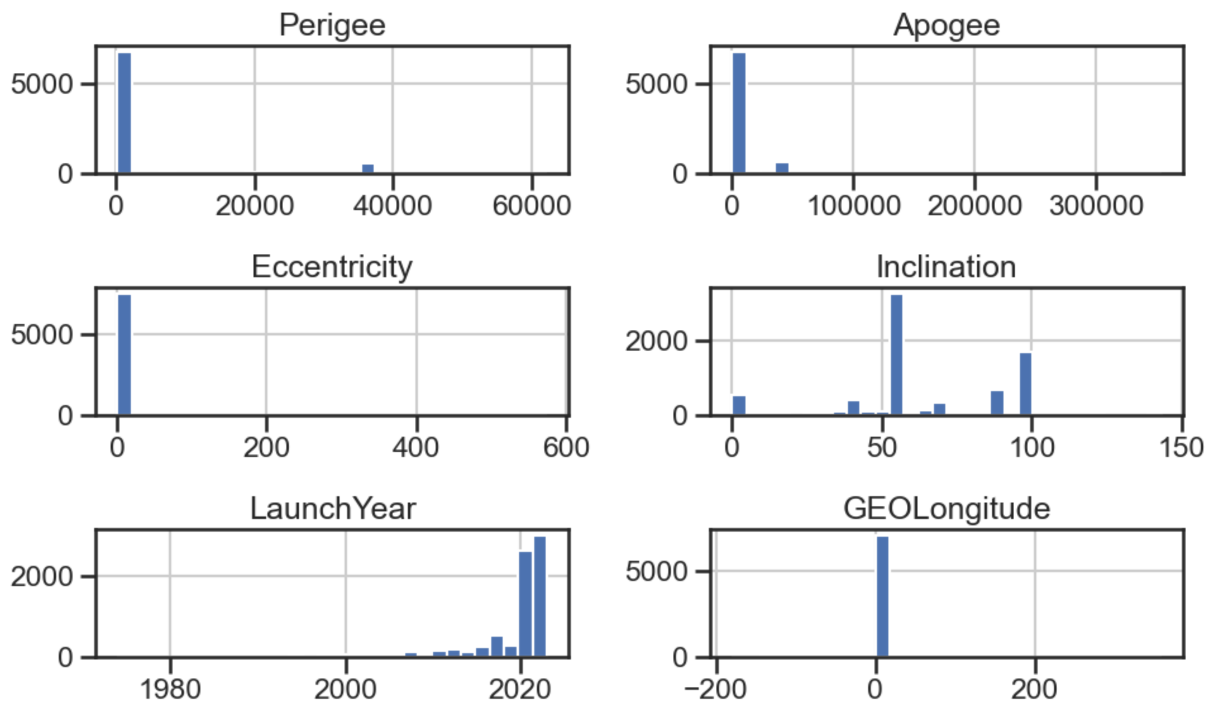
- **Confusion Matrix:** Strong diagonal dominance; limited confusion between science-related classes.
- **Pairplots & Countplots:** Clear separability in `OrbitClass`, `Inclination`, and `Apogee`.

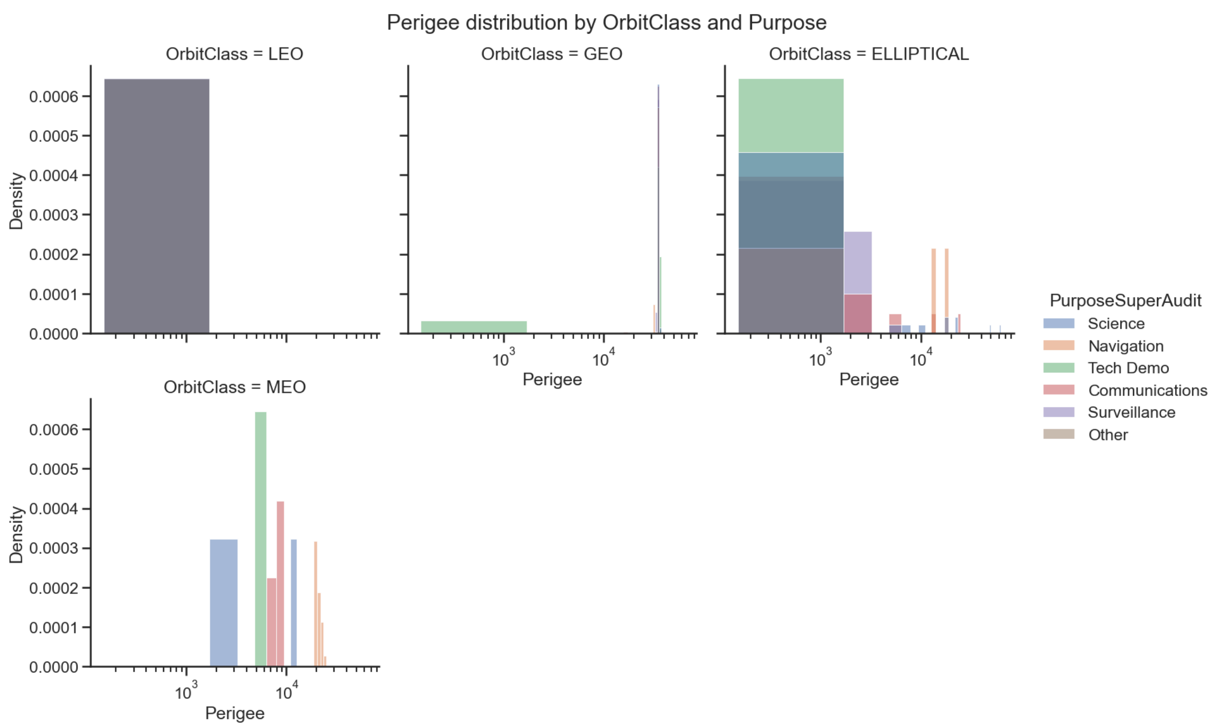
Bar chart shows how many satellites are in each Purpose bucket. Histograms show the spread of the numeric features (Perigee, Apogee, etc.).

In [71]:
```
# Class counts bar chart
dfm["PurposeSuperAudit"].value_counts().plot(kind="bar", rot=45)
```

```python
plt.title("Purpose (Super Buckets) — Count")
plt.ylabel("Count")
plt.tight_layout()
plt.show()

# Histograms for key numerics
dfm[num_cols].hist(bins=30, figsize=(10, 6))
plt.tight_layout()
plt.show()

if 'OrbitClass' in plot_df.columns:
    sns.displot(
        data=plot_df.dropna(subset=['Perigee']),
        x='Perigee', hue=target_col, col='OrbitClass',
        kind='hist', bins=40, multiple='layer', stat='density',
        common_bins=True, common_norm=False, col_wrap=3
    ).set(xscale='log')  # log helps for Perigee/Apogee
    plt.suptitle('Perigee distribution by OrbitClass and Purpose', y=1.02)
    plt.show()
```







In [ ]:

Discussion

Implications

- Orbital and physical parameters are powerful proxies for mission classification.
- Clear pattern: *Communications* satellites occupy **GEO**, while *Science* and *Tech Demo* favor **LEO / Sun-synchronous**.

- Ownership analysis revealed modern trends — e.g., Starlink's contribution dominates recent *Communications* entries.

Limitations

- Missing payload descriptors (sensor type, transponder details).
- Severe **class imbalance**—Communications dominates dataset.
- Dataset is static, omitting temporal behavior (orbital decay, end-of-life status).

Future Work

1. Integrate **payload and bus data** from open orbital catalogs.
2. Experiment with **tree ensembles** or **gradient boosting** for non-linear interactions.
3. Explore **hierarchical classification** (broad mission → sub-type).
4. Add time-series analysis (launch trends by year, orbit drift).

Summary

- Built a supervised ML model predicting satellite mission from orbital/physical data.
- Achieved **94% accuracy** using multinomial Logistic Regression.
- Demonstrated that simple linear models already capture strong mission–orbit relationships.
- Accuracy improved further through **category normalization** and **purpose reclassification** using domain knowledge.
- Future work: rebalance classes, add richer features, and explore non-linear models for higher-granularity prediction.
