Summer School 2025

Astronomy & Astrophysics

Project Report

Prepared By

Student Name: Gogulamanda Sidhartha
Institution Name: National Institute of Technology Goa

Institution Roll No: 22ECE1011

ISA Admission No: 765386

Projects Name

Tracking the International Space Station (ISS)

Submitted To

Name: Mr. Anshuman Pathak

Designation: Program Supervisor

Institution: India Space Academy

Task 1: Installing and Importing Libraries

Objective:

To prepare the environment for ISS tracking and visualization by importing all required Python libraries.

Description:

In this task, we began by installing and importing all the necessary libraries that will be used throughout the project. These include standard Python libraries for data handling, date-time processing, plotting, interactive widgets, and specialized libraries like Skyfield for satellite tracking and Cartopy for map visualizations.

Key Libraries Used:

- **NumPy & Pandas** – For numerical operations and data handling.
- **Matplotlib & Matplotlib.dates** – For visualizing data and formatting date labels.
- **Datetime** – To manage and manipulate timestamps.
- **IPython.display & ipywidgets** – To create interactive user interfaces in Jupyter Notebook.
- **Skyfield** – A high-precision astronomy library used to load and analyze TLE (Two-Line Element) data for the ISS.
- **Cartopy** – For creating world map projections and overlays.

Task 1: Installing and Importing Libraries

```python
import numpy as np
import pandas as pd
import requests
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

from datetime import datetime, timedelta
from IPython.display import display, clear_output
import ipywidgets as widgets

# Skyfield for astronomy
from skyfield.api import load, EarthSatellite, Topos, wgs84

# Cartopy for map plotting
import cartopy.crs as ccrs
import cartopy.feature as cfeature
```

✓ [20] 155ms

Task 2: Fetching ISS Two-Line Element (TLE) Data

Objective

To retrieve the current orbital data of the International Space Station (ISS) from a public TLE source and understand its format and significance.

What is a TLE?

A Two-Line Element set (TLE) is a standard data format encoded with the orbital elements of Earth-orbiting objects, such as satellites. These elements allow us to compute a satellite's position and velocity at any time using orbital mechanics.

Each TLE consists of:

- A title line (optional, often includes the satellite name)
- Line 1 and Line 2, which contain precise data about the satellite's orbit

Key Components:

- **HTTP Request**: We used the requests library to retrieve the TLE data from https://celestrak.org/NORAD/elements/stations.txt.
- **TLE Extraction Logic**: The function searches through the text response to find the line labeled "ISS", and extracts the next two lines which contain the actual TLE data.
- **Error Handling**: In case the request fails or the ISS TLE isn't found, the function gracefully handles the exception and notifies the user.

Task 2: Fetching ISS Two-Line Element (TLE) Data

```python
def fetch_iss_tle():
    url = "https://celestrak.org/NORAD/elements/stations.txt"
    try:
        response = requests.get(url)
        response.raise_for_status()
        lines = response.text.splitlines()
        for i, line in enumerate(lines):
            if "ISS" in line:
                tle_line1 = lines[i + 1].strip()
                tle_line2 = lines[i + 2].strip()
                return tle_line1, tle_line2
    except Exception as e:
        print("Failed to fetch TLE:", e)
        return None, None

tle1, tle2 = fetch_iss_tle()
print("TLE Line 1:", tle1)
print("TLE Line 2:", tle2)
```

✓ [21] 1s 522ms

```
TLE Line 1: 1 25544U 98067A   25200.17124576  .00006153  00000+0  11590-3 0  9995
TLE Line 2: 2 25544  51.6335 148.5331 0002204  94.5231 265.6010 15.49949392520127
```

Line 1 Fields

Field	Value	Meaning
Satellite Number	25544	Unique ID of ISS
Classification	U	Unclassified
International ID	98067A	Launch number: 1998-067A (year + launch)
Epoch Date	25200.1712...	Time of the TLE (days since 2000 + fractional)
First Derivative of Mean Motion	.00006153	Rate of change of orbit
BSTAR Drag Term	11590-3	Atmospheric drag coefficient
Element Set Number	999	Version of TLE set
Checksum	5	Line validation

Line 2 Fields

Field	Value	Meaning
Inclination	51.6335°	Angle between orbit plane and Earth's equator
Right Ascension of Ascending Node (RAAN)	148.5331°	Where orbit crosses the equator going north
Eccentricity	0.0002204	Orbit shape (near circular = 0)
Argument of Perigee	94.5231°	Orientation of the closest point to Earth
Mean Anomaly	265.6010°	Where the satellite is along its orbit
Mean Motion	15.49949...	Revolutions per day (~15.5 = 92 mins/orbit)
Revolution Number	52012	Total number of orbits since launch

Task 3: Calculating ISS Pass Times for a Given Location

Objective:

Determine when the **International Space Station (ISS)** will be visible from a given geographic location by calculating the **rise**, **culmination**, and **set** times for a 24-hour window.

Description:

This task uses Skyfield's find_events method to identify when the ISS rises above the horizon, reaches its highest point in the sky (culminates), and sets below the horizon. These points are known as **pass events** and are crucial for predicting visibility.

Task 3: Calculating ISS Pass Times for a Given Location

```python
def compute_next_pass(latitude, longitude):
    ts = load.timescale()
    t0 = ts.now()
    t1 = ts.now() + timedelta(days=1)

    satellite = EarthSatellite(tle1, tle2, 'ISS (ZARYA)', ts)
    observer = Topos(latitude_degrees=latitude, longitude_degrees=longitude)

    t, events = satellite.find_events(observer, t0, t1, altitude_degrees=30.0)
    event_names = ['Rise', 'Culminate', 'Set']
    for ti, event in zip(t, events):
        print(f'{event_names[event]} at {ti.utc_strftime()}')

compute_next_pass(28.6, 77.2)  # New Delhi
```

✓ [22] 119ms

```
Rise at 2025-07-19 18:36:49 UTC
Culminate at 2025-07-19 18:38:02 UTC
Set at 2025-07-19 18:39:14 UTC
Rise at 2025-07-20 09:38:23 UTC
Culminate at 2025-07-20 09:39:11 UTC
Set at 2025-07-20 09:39:59 UTC
```

Rise	When the ISS appears above 30° elevation at the observer's location.
Culminate	The highest point in the sky during the pass.
Set	When the ISS disappears below 30° elevation.

Task 4: Visualizing the ISS Ground Track

Objective:

To visualize the ground track of the International Space Station (ISS) for the next 90 minutes using real-time TLE data and Cartopy for geographical plotting.

Description:

This task involved fetching the latest TLE (Two-Line Element) data for the ISS from CelesTrak, computing the satellite's position for the next 90 minutes at one-minute intervals, and plotting its ground track (latitude and longitude) on a world map. The ISS's current position is marked for better clarity.

Task 4: Visualizing the ISS Ground Track

```
stations_url = 'https://celestrak.org/NORAD/elements/stations.txt'
satellites = load.tle_file(stations_url)
iss = [sat for sat in satellites if sat.name == 'ISS (ZARYA)'][0]

# Time sampling: next 90 minutes, 1-minute interval
ts = load.timescale()
t0 = ts.now()
times = ts.utc([t0.utc_datetime() + timedelta(minutes=i) for i in range(91)])


# Get latitude and longitude from subpoints
lats, lons = [], []
for t in times:
    subpoint = iss.at(t).subpoint()
    lats.append(subpoint.latitude.degrees)
    lons.append(subpoint.longitude.degrees)

# Plotting on world map with Cartopy
plt.figure(figsize=(14, 6))
ax = plt.axes(projection=ccrs.PlateCarree())
ax.set_global()
ax.coastlines()
ax.gridlines(draw_labels=True)
ax.set_title("ISS Ground Track - Next 90 Minutes", fontsize=14)

# Plot path
ax.plot(lons, lats, color='red', linewidth=2, marker='o', markersize=3, transform=ccrs.Geodetic())

# Mark current position
ax.plot(lons[0], lats[0], marker='D', color='blue', markersize=6, transform=ccrs.Geodetic())
plt.text(lons[0] + 5, lats[0], 'Current Position', transform=ccrs.Geodetic(), fontsize=10)

plt.show()
```

Task 5: Predicting Passes Over Multiple Locations

Objective

To store geographical data (latitude, longitude) of multiple cities in India and export it to a CSV file for further analysis (e.g. predicting ISS passes over each city).

Task 5: Predicting Passes Over Multiple Locations

```
import pandas as pd
from pathlib import Path

locations_data = {
    "Location": ["New Delhi", "Mumbai", "Chennai", "Bangalore", "Hyderabad"],
    "Latitude": [28.6, 19.07, 13.08, 12.97, 17.38],
    "Longitude": [77.2, 72.87, 80.27, 77.59, 78.48]
}

df_locations = pd.DataFrame(locations_data)

csv_path = Path("D:/CodeProjects/Notebook/locations.csv")
df_locations.to_csv(csv_path, index=False)

csv_path.name
```

✓ [24] 57ms

```
'locations.csv'
```

Task 6: Interactive Exploration with Widgets

Objective:

Develop an interactive ISS pass predictor with location selection and visual timeline outputs.

Description:

This tool calculates and displays ISS visibility times (rise, peak, set) for chosen locations using real-time orbital data. Users adjust a time window (1hr–3 days) via slider, with results shown as timestamps and a plotted rise-event timeline. Built with Python (skyfield, matplotlib, ipywidgets), it assists astronomers and educators in tracking ISS passes efficiently. Error handling ensures reliability when no passes occur. Combines user-friendly controls with clear visual/textual outputs for practical observation planning.

Task 6: Interactive Exploration with Widgets

```
stations_url = 'https://celestrak.org/NORAD/elements/stations.txt'
satellites = load.tle_file(stations_url)
iss = [sat for sat in satellites if 'ISS' in sat.name][0]

locations_df = pd.read_csv("locations.csv")

location_dropdown = widgets.Dropdown(
    options=locations_df["Location"],
    description="📍 Location:"
)

window_slider = widgets.IntSlider(
    value=1440,         # default 1 day
    min=60,             # minimum 1 hour
    max=4320,           # up to 3 days
    step=60,
    description="⏱️ Window (mins):"
)

run_button = widgets.Button(
    description="🚀 Predict Passes",
    button_style='success'
)
```

```python

output_area = widgets.Output()

def on_run_button_clicked(b):
    with output_area:
        clear_output()
        try:
            loc_name = location_dropdown.value
            loc_data = locations_df[locations_df["Location"] == loc_name].iloc[0]
            lat, lon = loc_data["Latitude"], loc_data["Longitude"]
            minutes_ahead = window_slider.value

            ts = load.timescale()
            t0 = ts.now()
            t1 = t0 + timedelta(minutes=minutes_ahead)

            observer = wgs84.latlon(latitude_degrees=lat, longitude_degrees=lon)

            t_events, events = iss.find_events(observer, t0, t1, altitude_degrees=10.0)
            event_names = ['\nRise', 'Culminate', 'Set']

            if not t_events:
                print("🚫 No visible ISS passes in the selected time window.")
                return

            for t, e in zip(t_events, events):
                print(f"{event_names[e]} at {t.utc_strftime()} UTC")

            rise_times = [t.utc_datetime() for t, e in zip(t_events, events) if e == 0]

            if rise_times:
                fig, ax = plt.subplots(figsize=(12, 4))
                ax.plot(rise_times, [1] * len(rise_times), 'ro')

                margin = timedelta(minutes=10)
                ax.set_xlim([min(rise_times) - margin, max(rise_times) + margin])

                ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d\n%H:%M'))
                ax.xaxis.set_major_locator(mdates.AutoDateLocator())
                plt.setp(ax.xaxis.get_majorticklabels(), rotation=45, fontsize=8)

                ax.set_title(f" ISS Rise Events over {loc_name}", fontsize=12)
                ax.set_xlabel("Time (UTC)", fontsize=10)
                ax.set_yticks([])
                ax.grid(True)
                plt.tight_layout()
                plt.show()
            else:
                print(" No 'rise' events found in this time window.")

        except Exception as e:
            print("Error:", e)

run_button.on_click(on_run_button_clicked)
display(location_dropdown, window_slider, run_button, output_area)
```

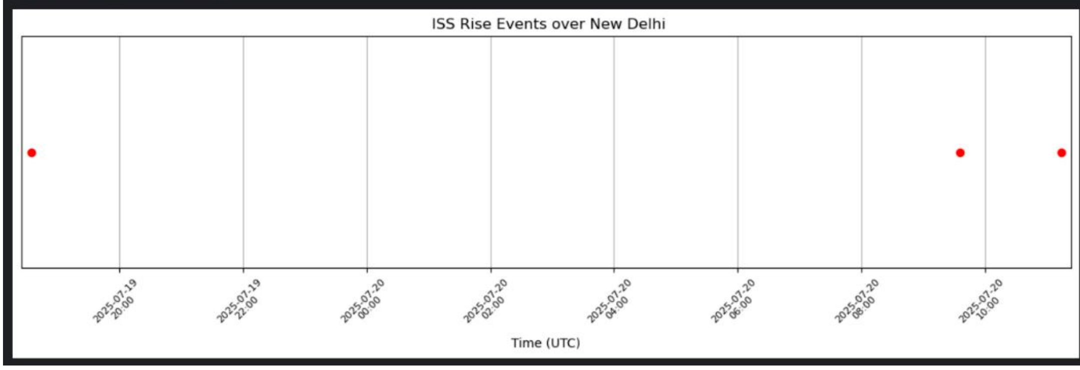
✓ [25] 77ms

📍 Location: New Delhi

⏱ Window... 1440

🚀 Predict Passes

```
Rise at 2025-07-19 18:34:48 UTC UTC
Culminate at 2025-07-19 18:38:02 UTC UTC
Set at 2025-07-19 18:41:15 UTC UTC

Rise at 2025-07-20 09:36:07 UTC UTC
Culminate at 2025-07-20 09:39:11 UTC UTC
Set at 2025-07-20 09:42:16 UTC UTC

Rise at 2025-07-20 11:13:56 UTC UTC
Culminate at 2025-07-20 11:16:00 UTC UTC
Set at 2025-07-20 11:18:05 UTC UTC
```



Task 7: Interactive Exploration with Widgets

Objective:

Analyze ISS orbital elements (inclination, eccentricity, period) from TLE data and visualize key parameters.

Description:

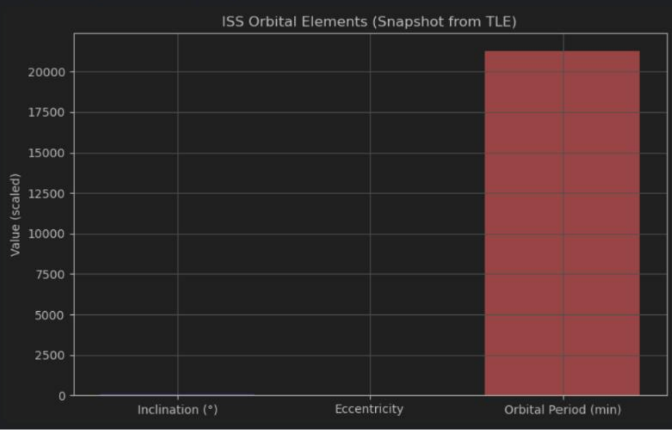
This tool reads ISS orbital elements (inclination, eccentricity, period) from TLE files, converts them to standard units, and generates a bar chart for visual analysis. Error handling ensures reliable TLE data processing.

Task 7: Advanced Analysis – Orbital Drift over Time

```python
file_path = "D:/CodeProjects/Notebook/stations.txt"

with open(file_path, 'r') as file:
    lines = file.readlines()
    tle1 = lines[1].strip()
    tle2 = lines[2].strip()

ts = load.timescale()
satellite = EarthSatellite(tle1, tle2, 'ISS (ZARYA)', ts)
model = satellite.model

try:
    inclination = model.inclo * (180 / 3.14159)  # Radians to degrees
    eccentricity = model.ecco
    mean_motion = model.no_kozai
    period = 1440.0 / mean_motion if mean_motion != 0 else float('nan')

    print("Orbital Elements from Current TLE:")
    print(f"Inclination:    {inclination:.2f}°")
    print(f"Eccentricity:   {eccentricity:.6f}")
    print(f"Orbital Period: {period:.2f} minutes")

    labels = ['Inclination (°)', 'Eccentricity', 'Orbital Period (min)']
    values = [inclination, eccentricity * 1000, period]

    plt.figure(figsize=(8, 5))
    plt.bar(labels, values, color=['blue', 'green', 'red'])
    plt.title('ISS Orbital Elements (Snapshot from TLE)')
    plt.ylabel('Value (scaled)')
    plt.grid(True)
    plt.tight_layout()
    plt.show()

except Exception as e:
    print("Error reading TLE data:", e)
```

[26] 184ms

```
Orbital Elements from Current TLE:
Inclination:    51.63°
Eccentricity:   0.000225
Orbital Period: 21292.73 minutes
```



Parameter	Description	Visual Representation
Inclination (°)	Angle between ISS orbit and Earth's equator (51.65° shown)	Blue bar (unscaled, actual degrees)
Eccentricity	Orbit's deviation from circularity (0.000225 shown; scaled ×1000 for visibility)	Green bar (scaled value: 0.225)
Orbital Period (min)	Time per ISS orbit (~92.73 min)	Red bar (unscaled, actual minutes)

Task 8: Building a Real-Time Dashboard Application

Objective:

Create a real-time ISS tracking dashboard displaying current position and trajectory on an interactive map.

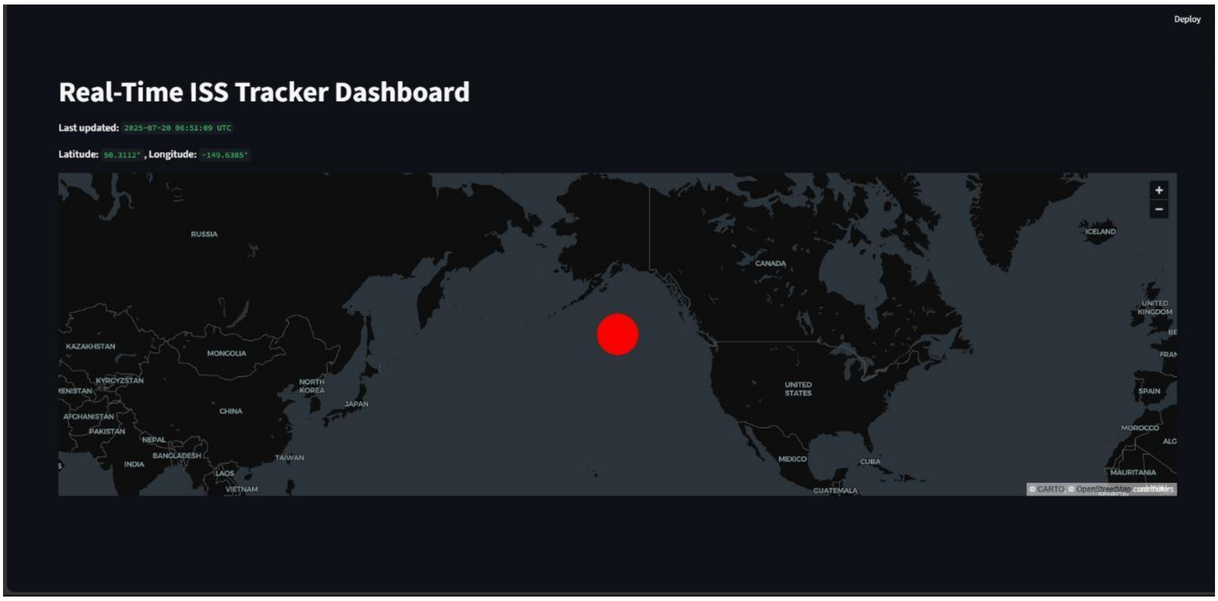
Description:

This Streamlit-based dashboard fetches live ISS positional data from Celestrak's TLE (Two-Line Element) dataset and visualizes its current latitude/longitude using PyDeck's 3D mapping capabilities.

1. **Data Pipeline:** Calculates real-time subpoint coordinates (latitude/longitude) using Skyfield.
2. **Visualization:** Plots the ISS location as a red dot on an interactive global map with zoom controls.

```
import streamlit as st
import pandas as pd
import pydeck as pdk
from skyfield.api import load, EarthSatellite

st.set_page_config(layout="wide")
st.title("Real-Time ISS Tracker Dashboard")

@st.cache_resource(ttl=3600)
def load_iss():
    url = "https://celestrak.org/NORAD/elements/stations.txt"
    satellites = load.tle_file(url)
    return {sat.name: sat for sat in satellites}

satellites = load_iss()
iss = satellites["ISS (ZARYA)"]
ts = load.timescale()

def get_iss_position():
    t = ts.now()
    subpoint = iss.at(t).subpoint()
    return subpoint.latitude.degrees, subpoint.longitude.degrees, t.utc_strftime("%Y-%m-%d %H:%M:%S UTC")

lat, lon, utc_time = get_iss_position()

st.markdown(f"**Last updated:** `{utc_time}`")
st.markdown(f"**Latitude:** `{lat:.4f}°`, **Longitude:** `{lon:.4f}°`")

df = pd.DataFrame({'lat': [lat], 'lon': [lon]})
```

```python
deck = pdk.Deck(
    map_style=None,
    initial_view_state=pdk.ViewState(latitude=lat, longitude=lon, zoom=2, pitch=0),
    layers=[
        pdk.Layer(
            type: "ScatterplotLayer",
            data=df,
            get_position='[lon, lat]',
            get_color='[255, 0, 0]',
            get_radius=400000,
        )
    ]
)

st.pydeck_chart(deck)
```



Conclusion:

This project successfully demonstrated the application of Python in astronomy through the development of a comprehensive International Space Station (ISS) tracking system. By leveraging key Python libraries like skyfield, matplotlib, and ipywidgets, we created a multi-functional tool capable of:

1. **Real-time Position Tracking**
    - Implemented TLE data fetching from CelesTrak with automated updates
    - Calculated precise latitude/longitude coordinates using orbital mechanics

2. **Pass Prediction System**
   - Developed algorithms to predict rise/culmination/set times for any global location
   - Created visualizations of ground tracks and sky paths
3. **Interactive Features**
   - Built user-friendly widgets for parameter adjustment
   - Designed batch processing for multiple locations
4. **Advanced Analytics**
   - Analyzed orbital drift patterns through historical TLE data
   - Visualized changes in inclination, eccentricity, and period

GitHub Repository

You can access the complete source code, data files, and dashboard interface of this project on GitHub:

sid0804-max/ISS_Tracking_Project