# TROPICAL TRENDSETTERS


## Robin Mikeal and Jef Hinton


**NOTES:**

- CLEAN DATA PREPARED IN LAST SUBMISSION
- DATA SELECTED ONLY FROM THE TAMPA RADIUS

In [56]:
```
#pip install folium
#pip install geopandas folium  #install in terminal or command line
#pip install geopy #install in terminal or command line
```

In [1]:
```
#import modules
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import folium
import geopandas as gpd
from branca.element import Element
import math
```

In [2]:
```
#import clean dataset
df = pd.read_csv("GEO557Tropical_Storm_Dataset_CLEAN.csv")
# import, get info and head to prove data exists.
print(df.info())
df.head(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Year      2240 non-null   int64  
 1   Name      2240 non-null   object 
 2   BASIN     0 non-null      float64
 3   ISO_TIME  2240 non-null   object 
 4   NATURE    2240 non-null   object 
 5   LAT       2240 non-null   float64
 6   LON       2240 non-null   float64
 7   WMO WIND  1180 non-null   float64
 8   WMO PRES  1180 non-null   float64
 9   USA WIND  2240 non-null   int64  
 10  USA PRES  2240 non-null   int64  
dtypes: float64(5), int64(3), object(3)
memory usage: 192.6+ KB
None
```

Out[2]:

| | Year | Name | BASIN | ISO_TIME | NATURE | LAT | LON | WMO WIND | WMO PRES | USA WIND | USA PRES |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2023 | IDALIA | NaN | 2023-08-26 12:00:00 | TS | 20.8 | -86.1 | 25.0 | 1006.0 | 25 | 1006 |
| **1** | 2023 | IDALIA | NaN | 2023-08-26 15:00:00 | TS | 21.1 | -86.1 | NaN | NaN | 25 | 1006 |
| **2** | 2023 | IDALIA | NaN | 2023-08-26 18:00:00 | TS | 21.3 | -86.2 | 25.0 | 1006.0 | 25 | 1006 |
| **3** | 2023 | IDALIA | NaN | 2023-08-26 21:00:00 | TS | 21.3 | -86.3 | NaN | NaN | 28 | 1005 |
| **4** | 2023 | IDALIA | NaN | 2023-08-27 00:00:00 | TS | 21.1 | -86.4 | 30.0 | 1004.0 | 30 | 1004 |
| **5** | 2023 | IDALIA | NaN | 2023-08-27 03:00:00 | TS | 20.8 | -86.7 | NaN | NaN | 30 | 1003 |
| **6** | 2023 | IDALIA | NaN | 2023-08-27 06:00:00 | TS | 20.5 | -86.8 | 30.0 | 1002.0 | 30 | 1002 |
| **7** | 2023 | IDALIA | NaN | 2023-08-27 09:00:00 | TS | 20.2 | -86.6 | NaN | NaN | 33 | 1001 |
| **8** | 2023 | IDALIA | NaN | 2023-08-27 12:00:00 | TS | 19.9 | -86.3 | 35.0 | 999.0 | 35 | 999 |
| **9** | 2023 | IDALIA | NaN | 2023-08-27 15:00:00 | TS | 19.9 | -86.0 | NaN | NaN | 38 | 998 |

In [3]:

```
#import Milton overlay
Milton = pd.read_csv("MILTON_AL142024_pts.csv")
# import, get info and head to prove data exists.
print(Milton.info())
Milton.head(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22 entries, 0 to 21
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   STORMNAME  22 non-null     object
 1   DTG        22 non-null     int64
 2   YEAR       22 non-null     int64
 3   MONTH      22 non-null     int64
 4   DAY        22 non-null     int64
 5   HHMM       22 non-null     int64
 6   MSLP       22 non-null     int64
 7   BASIN      22 non-null     object
 8   STORMNUM   22 non-null     int64
 9   STORMTYPE  22 non-null     object
 10  INTENSITY  22 non-null     int64
 11  SS         22 non-null     int64
 12  LAT        22 non-null     int64
 13  LON        22 non-null     int64
dtypes: int64(11), object(3)
memory usage: 2.5+ KB
None
```

Out[3]:

| | STORMNAME | DTG | YEAR | MONTH | DAY | HHMM | MSLP | BASIN | STORMNUM | STORMTYP |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | FOURTEEN | 2024100512 | 2024 | 10 | 5 | 1200 | 1007 | al | 14 | T |
| **1** | MILTON | 2024100518 | 2024 | 10 | 5 | 1800 | 1006 | al | 14 | T |
| **2** | MILTON | 2024100600 | 2024 | 10 | 6 | 0 | 1006 | al | 14 | T |
| **3** | MILTON | 2024100606 | 2024 | 10 | 6 | 600 | 1000 | al | 14 | T |
| **4** | MILTON | 2024100612 | 2024 | 10 | 6 | 1200 | 991 | al | 14 | T |
| **5** | MILTON | 2024100618 | 2024 | 10 | 6 | 1800 | 987 | al | 14 | H |
| **6** | MILTON | 2024100700 | 2024 | 10 | 7 | 0 | 981 | al | 14 | H |
| **7** | MILTON | 2024100706 | 2024 | 10 | 7 | 600 | 972 | al | 14 | H |
| **8** | MILTON | 2024100712 | 2024 | 10 | 7 | 1200 | 943 | al | 14 | H |
| **9** | MILTON | 2024100718 | 2024 | 10 | 7 | 1800 | 909 | al | 14 | H |

In [4]:

```python
filtered_df = df[(df['USA WIND'] >= 74) & (df['USA WIND'] <= 95)]

filtered_df
```

Out[4]:

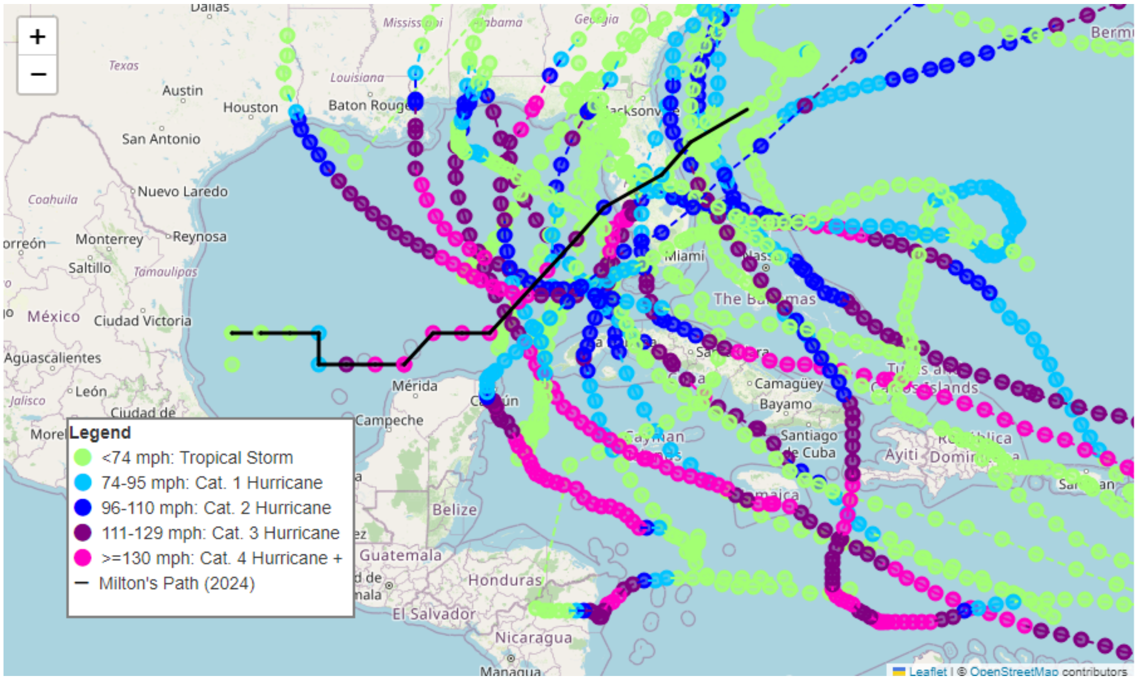
| | Year | Name | BASIN | ISO_TIME | NATURE | LAT | LON | WMO WIND | WMO PRES | USA WIND | USA PRES |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **25** | 2023 | IDALIA | NaN | 2023-08-29 15:00:00 | TS | 24.5 | -84.8 | NaN | NaN | 75 | 976 |
| **26** | 2023 | IDALIA | NaN | 2023-08-29 18:00:00 | TS | 25.3 | -84.8 | 80.0 | 973.0 | 80 | 973 |
| **27** | 2023 | IDALIA | NaN | 2023-08-29 21:00:00 | TS | 26.1 | -84.8 | NaN | NaN | 85 | 969 |
| **28** | 2023 | IDALIA | NaN | 2023-08-30 00:00:00 | TS | 26.9 | -84.7 | 90.0 | 965.0 | 90 | 965 |
| **34** | 2023 | IDALIA | NaN | 2023-08-30 15:00:00 | TS | 30.9 | -82.8 | NaN | NaN | 80 | 968 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2094** | 2004 | IVAN | NaN | 2004-09-06 21:00:00 | TS | 11.3 | -55.3 | NaN | NaN | 90 | 967 |
| **2095** | 2004 | IVAN | NaN | 2004-09-07 00:00:00 | TS | 11.2 | -56.1 | 90.0 | 964.0 | 90 | 964 |
| **2096** | 2004 | IVAN | NaN | 2004-09-07 03:00:00 | TS | 11.2 | -57.0 | NaN | NaN | 93 | 965 |
| **2097** | 2004 | IVAN | NaN | 2004-09-07 06:00:00 | TS | 11.3 | -57.8 | 95.0 | 965.0 | 95 | 965 |
| **2175** | 2004 | IVAN | NaN | 2004-09-16 09:00:00 | TS | 30.7 | -87.8 | NaN | NaN | 90 | 954 |

254 rows × 11 columns

In [27]:
```
# Function to determine the color based on wind speed
def get_color(wind_speed):
    if wind_speed < 74:
        return '#A3FF73'   # Green
    elif 74 <= wind_speed < 95:
        return '#00C5FF'   # Turquoise Blue
    elif 95 <= wind_speed < 110:
        return 'blue'
    elif 110 <= wind_speed < 129:
        return 'purple'
    elif wind_speed >= 129:
        return '#FF00C5'   # Electric Pink

# Filter the DataFrame to include only the hurricane data with wind >= 40 mph
hurricane_path = df[df['USA WIND'] >= 40].dropna(subset=['LAT', 'LON'])

# Create a map centered around Florida with OpenStreetMap tiles
m = folium.Map(
    location=[hurricane_path['LAT'].mean(), hurricane_path['LON'].mean()],
    tiles='OpenStreetMap',
    zoom_start=4
)

# Group by 'Name' to connect points of the same hurricane
for name, group in hurricane_path.groupby('Name'):
    previous_location = None
    previous_color = None

    # Add markers for each point in the group
    for _, row in group.iterrows():
        location = [row['LAT'], row['LON']]
        popup = f"{row['Name']}<br>Wind: {row['USA WIND']} mph<br>Pressure: {row['USA

        # Get the color based on the wind speed
        color = get_color(row['USA WIND'])

        # Add a circle marker for each data point
        folium.CircleMarker(
            location=location,
            radius=5,
            color=color,
            fill=True,
            fill_color=color,
            fill_opacity=0.6,
            popup=popup
        ).add_to(m)

        # Draw a polyline from the previous point to the current point, if a previous
        if previous_location is not None:
            folium.PolyLine(
                locations=[previous_location, location],
                color=previous_color,  # Set line color to previous point's color
                weight=2,
                dash_array='5, 5'  # Dashed line effect
            ).add_to(m)

        # Update the previous point information
        previous_location = location
        previous_color = color
```

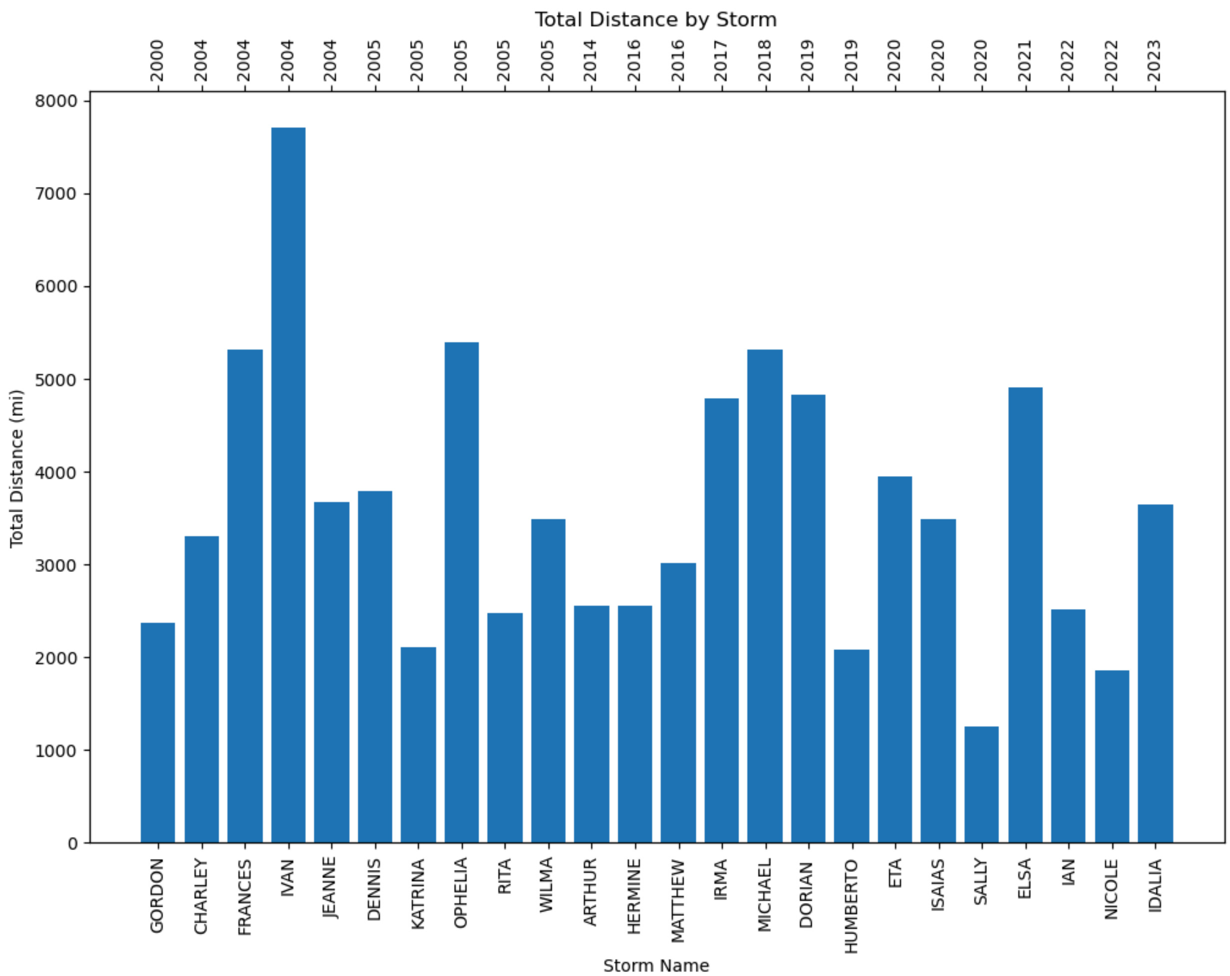
```
#TEST CSV Milton
def add_marker(map_obj, location, popup, color):
    folium.CircleMarker(
        location=location,
        radius=5,
        color=color,
        fill=True,
        fill_color=color,
        fill_opacity=0.6,
        popup=popup
    ).add_to(map_obj)

for name, group in Milton.groupby('STORMNAME'):
    previous_location = None
    previous_color = None

    for _, row in group.iterrows():
        location = [row['LAT'], row['LON']]
        popup = f"{row['STORMNAME']}<br>Wind: {row['INTENSITY']} mph<br>Pressure: {row
        color = get_color(row['INTENSITY'])

        add_marker(m, location, popup, color)

        if previous_location is not None:
            folium.PolyLine(
                locations=[previous_location, location],
                color='black',  # Changed line color to black
                weight=3,).add_to(m)

        previous_location = location
        previous_color = color

# Add legend for data points
legend_html = '''
<div style="position: fixed;
     bottom: 50px; left: 50px; width: 230px; height: 160px;
     border:2px solid grey; z-index:9999; font-size:10x;
     background-color:white;
     padding: 10x
     ">
     <b>Legend</b><br>
      <i class="fa fa-circle" style="color:#A3FF73"></i>  <74 mph: Tropical
      <i class="fa fa-circle" style="color:#00C5FF"></i>  74-95 mph: Cat. 1
      <i class="fa fa-circle" style="color:blue"></i>  96-110 mph: Cat. 2 Hu
      <i class="fa fa-circle" style="color:purple"></i>  111-129 mph: Cat. 3
      <i class="fa fa-circle" style="color:#FF00C5"></i>  >=130 mph: Cat. 4
      <i class="fa fa-minus" style="color:black"></i>  Milton's Path (2024)
</div>
'''
m.get_root().html.add_child(folium.Element(legend_html))

legend = Element(legend_html)
m.get_root().add_child(legend)

# Display the map
m
```

Out[27]: Make this Notebook Trusted to load map: File -> Trust Notebook




In [6]:
```python
# Function to calculate distance between two points (latitude and longitude)
from geopy.distance import geodesic

# Function to calculate distance between two points in miles
def calculate_distance(point1, point2):
    return geodesic(point1, point2).miles

# Calculate the sum of distances by group and sort by Year
grouped = df.groupby(['Year', 'Name'])
distances = {}

for (year, name), group in grouped:
    total_distance = 0
    points = list(zip(group['LAT'], group['LON']))
    for i in range(len(points) - 1):
        total_distance += calculate_distance(points[i], points[i + 1])
    distances[(year, name)] = total_distance

# Sort distances by Year
sorted_distances = dict(sorted(distances.items(), key=lambda item: item[0]))

# Create a bar chart with vertical x-axis labels and labels of Year and Name on bars
fig, ax1 = plt.subplots(figsize=(12, 8))

bars = ax1.bar(range(len(sorted_distances)), sorted_distances.values(), tick_label=[f'
ax1.set_xlabel('Storm Name')
ax1.set_ylabel('Total Distance (mi)')
ax1.set_title('Total Distance by Storm')
ax1.set_xticks(range(len(sorted_distances)))
ax1.set_xticklabels([f"{name}" for year, name in sorted_distances.keys()], rotation='v

# Add a second x-axis for the years
ax2 = ax1.twiny()
ax2.set_xlim(ax1.get_xlim())
ax2.set_xticks(range(len(sorted_distances)))
ax2.set_xticklabels([f"{year}" for year, name in sorted_distances.keys()], rotation='v
```

```
plt.show()
```




El Niño-Southern Oscillation (ENSO), a natural climate pattern that involves changes in the temperature of the Pacific Ocean and the atmosphere: El Niño: A warming of the ocean surface in the central and eastern tropical Pacific Ocean. This phase is characterized by reduced rainfall over Indonesia and increased rainfall over the central and eastern tropical Pacific Ocean. La Niña: A cooling of the ocean surface in the central and eastern tropical Pacific Ocean. This phase is characterized by stronger east to west surface winds. Southern Oscillation: The atmospheric counterpart to El Niño and La Niña (SOURCE: NOAA, 2024).

In [7]:
```
dfENSO = pd.read_csv("ENSO_Years.csv")
# import, get info and head to prove data exists.
print(dfENSO.info())
dfENSO.head(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 127 entries, 0 to 126
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Year    127 non-null    int64
 1   ENSO    127 non-null    object
dtypes: int64(1), object(1)
memory usage: 2.1+ KB
None
```

Out[7]:

| | Year | ENSO |
|---|---|---|
| 0 | 1896 | Neutral |
| 1 | 1897 | El Niño |
| 2 | 1898 | Neutral |
| 3 | 1899 | Neutral |
| 4 | 1900 | El Niño |
| 5 | 1901 | Neutral |
| 6 | 1902 | Neutral |
| 7 | 1903 | El Niño |
| 8 | 1904 | La Niña |
| 9 | 1905 | Neutral |

In [8]:

```
#merge ENSO Year Table with the Hurricane Path dataframe
df2 = pd.merge(df, dfENSO, on='Year')
print(df2)
```

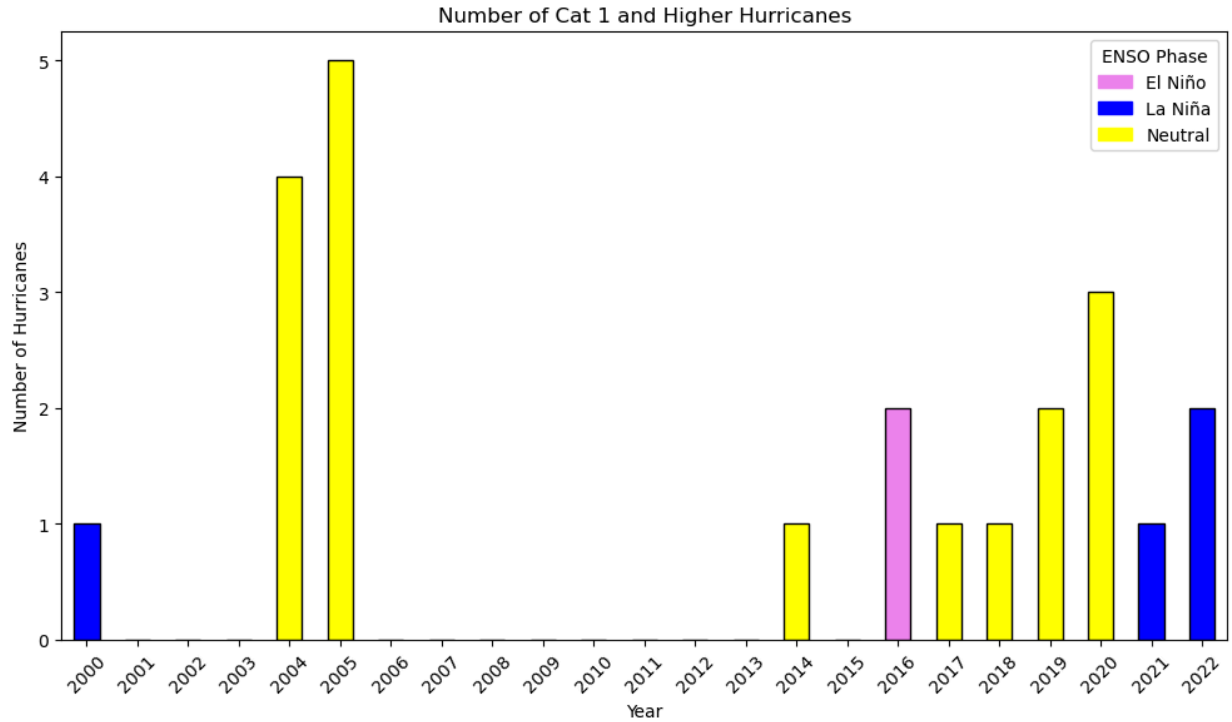
```
      Year    Name  BASIN             ISO_TIME NATURE   LAT   LON  WMO WIND  \
0     2022  NICOLE    NaN  2022-11-06 12:00:00     DS  20.6 -66.8      30.0
1     2022  NICOLE    NaN  2022-11-06 15:00:00     DS  21.5 -66.7       NaN
2     2022  NICOLE    NaN  2022-11-06 18:00:00     DS  22.4 -66.8      35.0
3     2022  NICOLE    NaN  2022-11-06 21:00:00     DS  23.2 -67.1       NaN
4     2022  NICOLE    NaN  2022-11-07 00:00:00     DS  23.9 -67.5      35.0
...    ...     ...    ...                  ...    ...   ...   ...       ...
2131  2000  GORDON    NaN  2000-09-20 18:00:00     ET  42.5 -67.2      30.0
2132  2000  GORDON    NaN  2000-09-20 21:00:00     ET  42.7 -66.1       NaN
2133  2000  GORDON    NaN  2000-09-21 00:00:00     ET  43.0 -65.0      30.0
2134  2000  GORDON    NaN  2000-09-21 03:00:00     ET  43.3 -64.0       NaN
2135  2000  GORDON    NaN  2000-09-21 06:00:00     ET  43.5 -63.0      30.0

      WMO PRES  USA WIND  USA PRES     ENSO
0       1005.0        30      1005  La Niña
1          NaN        33      1005  La Niña
2       1005.0        35      1005  La Niña
3          NaN        35      1005  La Niña
4       1005.0        35      1005  La Niña
...        ...       ...       ...      ...
2131    1005.0        30      1005  La Niña
2132       NaN        30      1005  La Niña
2133    1004.0        30      1004  La Niña
2134       NaN        30      1004  La Niña
2135    1003.0        30      1003  La Niña

[2136 rows x 12 columns]
```

In [9]:

```
#QA Check
# Filter records for Year 2005
#records_2005 = df2[df2['Year'] == 2005]
#records_2005

# Group by Year and count unique names
```

```
#grouped_unique_count = df2.groupby('Year')['Name'].nunique().reset_index(name='Unique
#grouped_unique_count
```

In [10]:
```
# Group by year and count unique storm names
storm_counts = df2.groupby('Year')['Name'].nunique()

# Create a range of years
all_years = pd.Series(range(df2['Year'].min(), df2['Year'].max() + 1), name='Year')

# Fill in missing years with 0
storm_counts = storm_counts.reindex(all_years, fill_value=0)

# Create a color map based on ENSO phases
enso_colors = {
    'El Niño': 'violet',
    'La Niña': 'blue',
    'Neutral': 'yellow'
}

# Map colors to the years based on ENSO phases
colors = df2.drop_duplicates('Year').set_index('Year')['ENSO'].reindex(all_years).map(

# Replace NaN values in colors with a default color (e.g., gray)
colors = colors.fillna('gray')

# Plot the data
plt.figure(figsize=(10, 6))
storm_counts.plot(kind='bar', color=colors, edgecolor='black')
# Label stuff
plt.xlabel('Year')
plt.ylabel('Number of Hurricanes')
plt.title('Number of Cat 1 and Higher Hurricanes')
plt.xticks(rotation=45)

# Add legend for colors
handles = [plt.Rectangle((0,0),1,1, color=color) for color in enso_colors.values()]
labels = enso_colors.keys()
plt.legend(handles, labels, title="ENSO Phase")

plt.tight_layout()
plt.show()
```



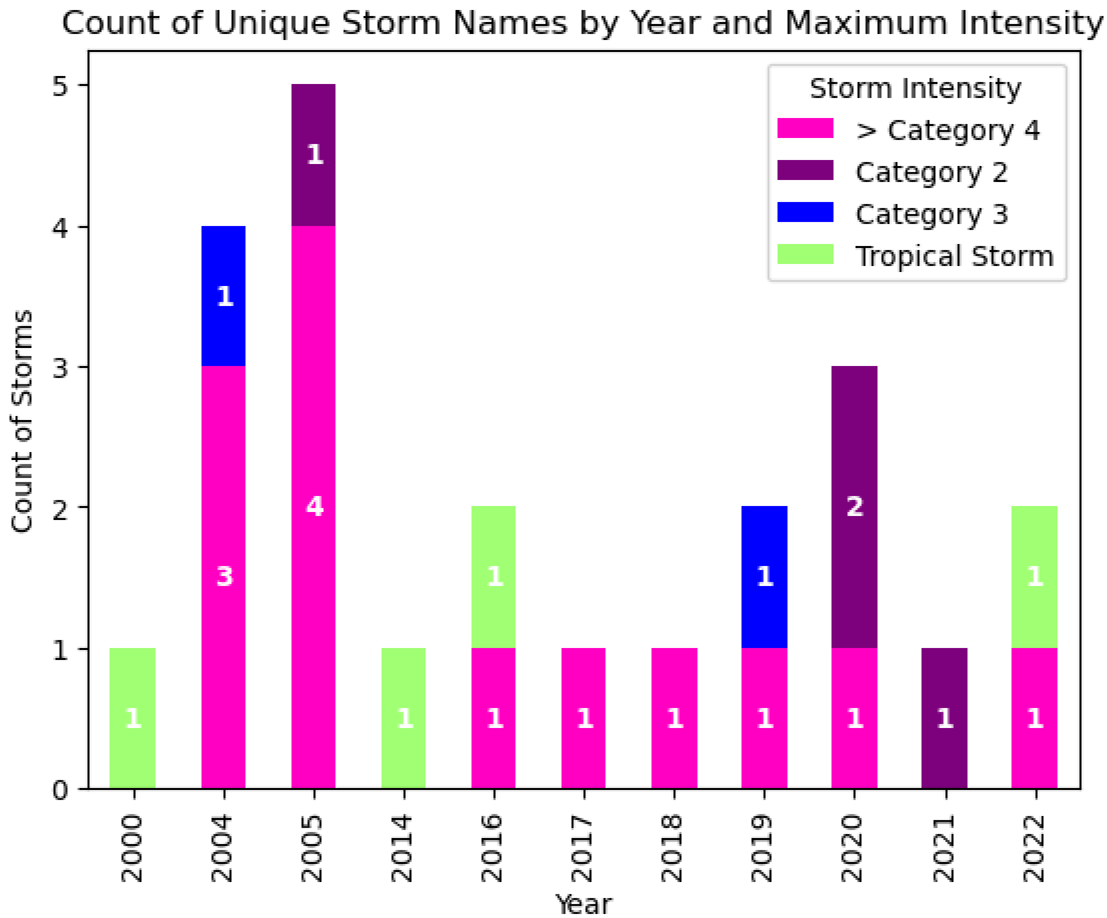

In [11]:

```
def category(wind_speed):
    if wind_speed < 74:
        return 'Tropical Storm'
    elif 74 >= wind_speed < 95:
        return 'Category 1'
    elif 95 >= wind_speed < 110:
        return 'Category 2'
    elif 110 >= wind_speed < 129:
        return 'Category 3'
    else:
        return '> Category 4'

# Apply the category function to create a new column
df2['Category'] = df2['USA WIND'].apply(category)

# Display the DataFrame with the new column
print(df2)
```

```
      Year    Name  BASIN             ISO_TIME NATURE   LAT   LON  WMO WIND  \
0     2022  NICOLE    NaN  2022-11-06 12:00:00     DS  20.6 -66.8      30.0
1     2022  NICOLE    NaN  2022-11-06 15:00:00     DS  21.5 -66.7       NaN
2     2022  NICOLE    NaN  2022-11-06 18:00:00     DS  22.4 -66.8      35.0
3     2022  NICOLE    NaN  2022-11-06 21:00:00     DS  23.2 -67.1       NaN
4     2022  NICOLE    NaN  2022-11-07 00:00:00     DS  23.9 -67.5      35.0
...    ...     ...    ...                  ...    ...   ...   ...       ...
2131  2000  GORDON    NaN  2000-09-20 18:00:00     ET  42.5 -67.2      30.0
2132  2000  GORDON    NaN  2000-09-20 21:00:00     ET  42.7 -66.1       NaN
2133  2000  GORDON    NaN  2000-09-21 00:00:00     ET  43.0 -65.0      30.0
2134  2000  GORDON    NaN  2000-09-21 03:00:00     ET  43.3 -64.0       NaN
2135  2000  GORDON    NaN  2000-09-21 06:00:00     ET  43.5 -63.0      30.0

      WMO PRES  USA WIND  USA PRES     ENSO        Category
0       1005.0        30      1005  La Niña  Tropical Storm
1          NaN        33      1005  La Niña  Tropical Storm
2       1005.0        35      1005  La Niña  Tropical Storm
3          NaN        35      1005  La Niña  Tropical Storm
4       1005.0        35      1005  La Niña  Tropical Storm
...        ...       ...       ...      ...             ...
2131    1005.0        30      1005  La Niña  Tropical Storm
2132       NaN        30      1005  La Niña  Tropical Storm
2133    1004.0        30      1004  La Niña  Tropical Storm
2134       NaN        30      1004  La Niña  Tropical Storm
2135    1003.0        30      1003  La Niña  Tropical Storm

[2136 rows x 13 columns]
```

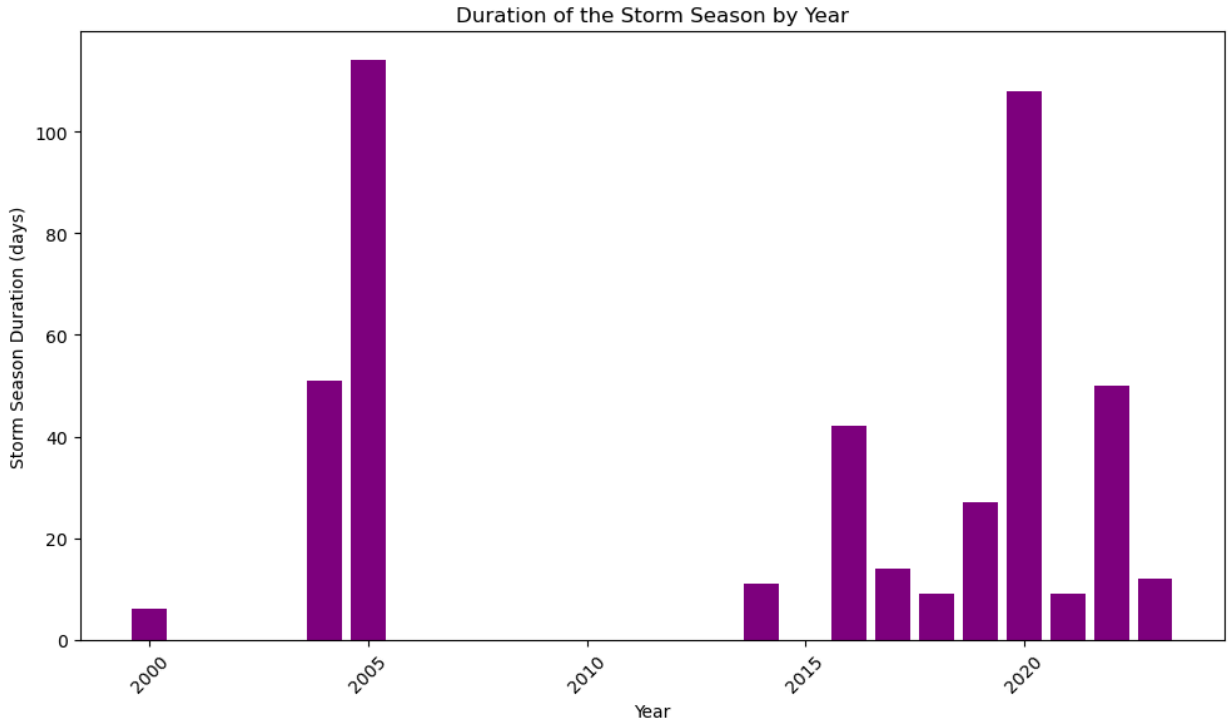
In [34]:

```python
# Find Max Wind by name, then group by year and category, then count unique storm name
# Step 1: Find the maximum wind speed by Name
max_wind_by_name = df2.groupby('Name')['USA WIND'].max().reset_index()

# Step 2: Merge this back with the original DataFrame to keep other columns
df2 = df2.merge(max_wind_by_name, on=['Name', 'USA WIND'])

# Step 3: Group by Year and Category, then count unique Names
result = df2.groupby(['Year', 'Category'])['Name'].nunique().unstack().fillna(0)

# Define custom color map
custom_colors = ['#FF00C5', 'purple', 'blue', '#A3FF73']

# Plotting the stacked bar chart with custom color map and white bold labels of counts
ax = result.plot(kind='bar', stacked=True, color=custom_colors)
plt.xlabel('Year')
plt.ylabel('Count of Storms')
plt.title('Count of Unique Storm Names by Year and Maximum Intensity')
plt.legend(title='Storm Intensity')

# Adding white bold labels of counts and removing zero labels
for container in ax.containers:
    labels = [int(v) if v > 0 else '' for v in container.datavalues]
    ax.bar_label(container, labels=labels, label_type='center', color='white', weight=

plt.show()
```



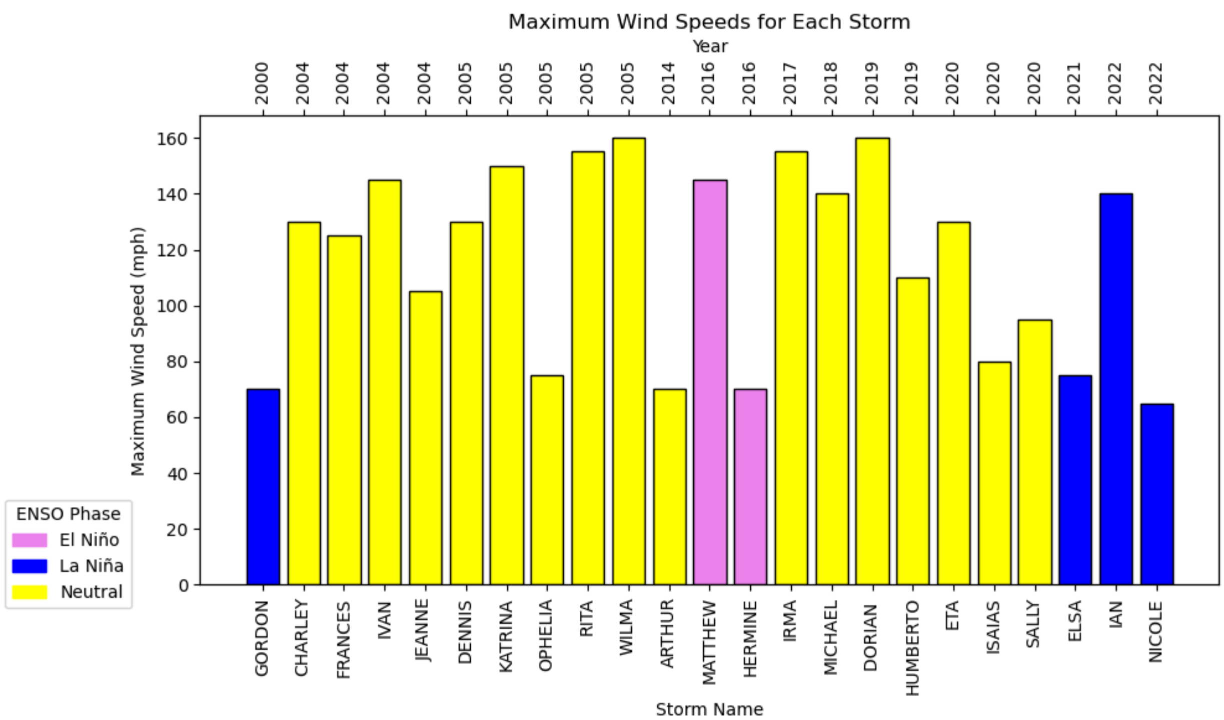

In [13]:

```
df['ISO_TIME'] = pd.to_datetime(df['ISO_TIME'])

# Group by year
season_durations = df.groupby('Year')['ISO_TIME'].agg(['min', 'max'])
season_durations['Duration'] = (season_durations['max'] - season_durations['min']).dt.

# Plot the duration of the storm season for each year
plt.figure(figsize=(10, 6))
plt.bar(season_durations.index, season_durations['Duration'], color='purple')
plt.xlabel('Year')
plt.ylabel('Storm Season Duration (days)')
plt.title('Duration of the Storm Season by Year')
plt.grid(False)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




In [14]:
```
# Group by storm name and get the maximum wind speed for each storm
max_wind_speeds = df2.groupby(['Year', 'Name'])['WMO WIND'].max().reset_index()

# Sort by year
max_wind_speeds = max_wind_speeds.sort_values(by='Year')

# Create a color map based on ENSO phases
enso_colors = {
    'El Niño': 'violet',
    'La Niña': 'blue',
    'Neutral': 'yellow'
}

# Map colors to the years based on ENSO phases
colors = df2.drop_duplicates('Year').set_index('Year')['ENSO'].map(enso_colors)

# Replace NaN values in colors with a default color (e.g., gray)
colors = colors.fillna('gray')

# Ensure the colors Series is aligned with the max_wind_speeds index
colors = colors.reindex(max_wind_speeds['Year']).values

# Plot the data
fig, ax1 = plt.subplots(figsize=(10, 6))

# Plot the maximum wind speeds
bars = ax1.bar(max_wind_speeds['Name'], max_wind_speeds['WMO WIND'], color=colors, edg

# Label stuff for the first axis
ax1.set_xlabel('Storm Name')
ax1.set_ylabel('Maximum Wind Speed (mph)')
ax1.set_title('Maximum Wind Speeds for Each Storm')
ax1.tick_params(axis='x', rotation=90)

# Add legend for colors
handles = [plt.Rectangle((0,0),1,1, color=color) for color in enso_colors.values()]
```

```
labels = enso_colors.keys()
plt.legend(handles, labels, title="ENSO Phase", loc='upper left', bbox_to_anchor=(-0.2

# Create a second x-axis to show the year labels
ax2 = ax1.twiny()

# Set the second x-axis limits to match the first x-axis
ax2.set_xlim(ax1.get_xlim())

# Set the second x-axis ticks and labels to show the years
ax2.set_xticks(range(len(max_wind_speeds)))
ax2.set_xticklabels(max_wind_speeds['Year'], rotation=90)

# Set the second x-axis label
ax2.set_xlabel('Year')

plt.tight_layout()
plt.show()
```



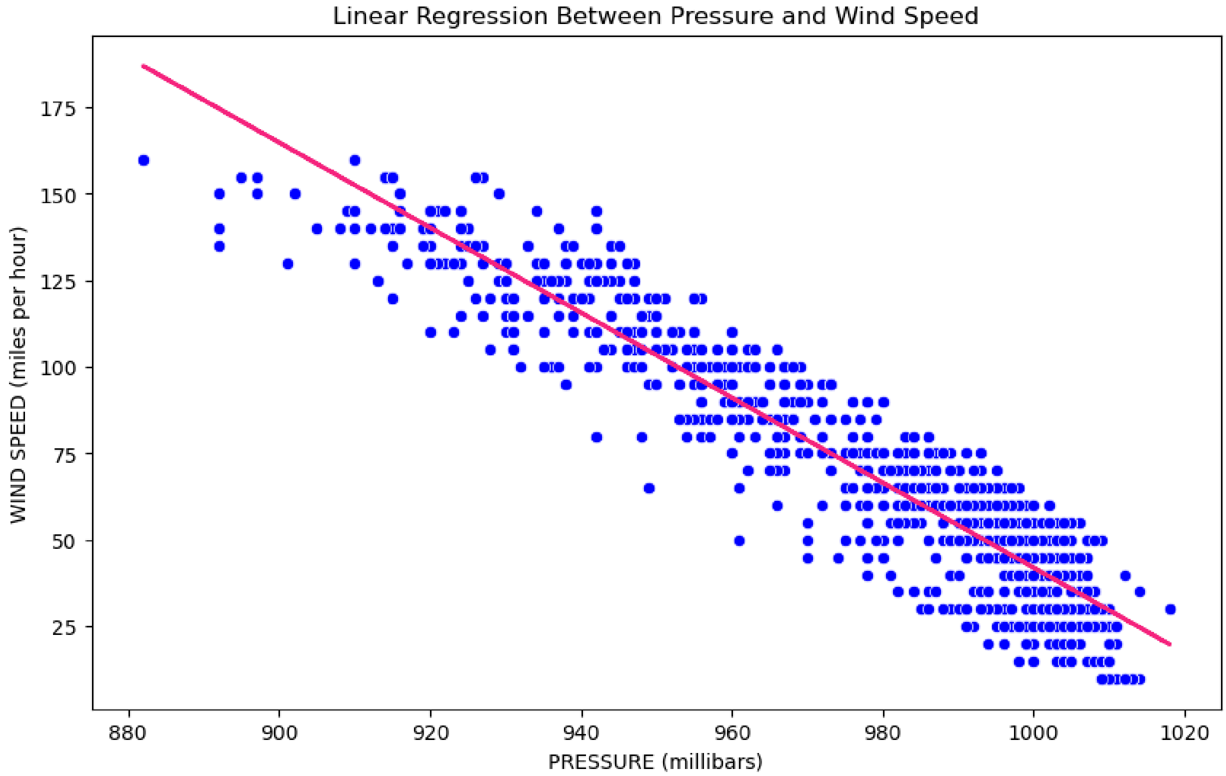

In [15]:
```
import seaborn as sns
from sklearn.linear_model import LinearRegression
```

In [16]:
```
nans_in_columns = df2.isna().sum()
print(nans_in_columns)
```

```
Year           0
Name           0
BASIN       2136
ISO_TIME       0
NATURE         0
LAT            0
LON            0
WMO WIND    1010
WMO PRES    1010
USA WIND       0
USA PRES       0
ENSO           0
Category       0
dtype: int64
```

In [17]:
```
df2 = df2.drop(columns=['BASIN'])
```

In [18]:
```
### Linear Regression of Pressure and Wind

# Drop rows with NaN values
df3 = df2.dropna(subset=['WMO WIND', 'WMO PRES'])

# Extract the relevant columns
X = df3[['WMO PRES']]
y = df3['WMO WIND']

# Create and fit the linear regression model
model = LinearRegression()
model.fit(X, y)

# Predict values
y_pred = model.predict(X)

# Plot the data and the regression line
plt.figure(figsize=(10, 6))
sns.scatterplot(x='WMO PRES', y='WMO WIND', data=df3, color='blue')
plt.plot(df3['WMO PRES'], y_pred, color='#F62681', linewidth=2)
plt.xlabel('PRESSURE (millibars)')
plt.ylabel('WIND SPEED (miles per hour)')
plt.title('Linear Regression Between Pressure and Wind Speed')
plt.show()
```


Linear Regression Between Pressure and Wind Speed
WIND SPEED (miles per hour)
175
150
125
100
75
50
25
880
900
920
940
960
980
1000
1020
PRESSURE (millibars)

In [ ]:

In [ ]: