Data Visualizations

Let's import the cleaned dataset

In [12]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.patches as mpatches


df = pd.read_csv('data/AviationData_CLEAN.csv', index_col=0, low_memor
df.head()
```

Out[12]:

Event.Id	Investigation.Type	Event.Date	Injury.Severity	Aircraft.damage	Make	Mod
20020909X01562	Accident	1982-01-01	Non-Fatal	Substantial	cessna	14
20020909X01559	Accident	1982-01-01	Non-Fatal	Substantial	piper	pa2816
20020909X01558	Accident	1982-01-01	Non-Fatal	Substantial	beech	v35
20020917X02134	Accident	1982-01-02	Fatal(1)	Destroyed	cessna	r172
20020917X02119	Accident	1982-01-02	Fatal(1)	Destroyed	navion	

In [17]:
```
df = df.dropna()
```

In [20]:
```
df['Event.Date'] = pd.to_datetime(df['Event.Date'], format='%Y-%m-%d')
```

In [21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 53266 entries, 20020909X01562 to 20221230106513
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype         
---  ------                  --------------  -----         
 0   Investigation.Type      53266 non-null  object        
 1   Event.Date              53266 non-null  datetime64[ns]
 2   Injury.Severity         53266 non-null  object        
 3   Aircraft.damage         53266 non-null  object        
 4   Make                    53266 non-null  object        
 5   Model                   53266 non-null  object        
 6   Number.of.Engines       53266 non-null  float64       
 7   Engine.Type             53266 non-null  object        
 8   Purpose.of.flight       53266 non-null  object        
 9   Total.Fatal.Injuries    53266 non-null  float64       
 10  Total.Serious.Injuries  53266 non-null  float64       
 11  Total.Minor.Injuries    53266 non-null  float64       
 12  Total.Uninjured         53266 non-null  float64       
 13  Weather.Condition       53266 non-null  object        
 14  Broad.phase.of.flight   53266 non-null  object        
 15  Report.Status           53266 non-null  object        
 16  Maker_Model             53266 non-null  object        
 17  General_Maker_Model     53266 non-null  object        
 18  Total_On_Board          53266 non-null  float64       
dtypes: datetime64[ns](1), float64(6), object(12)
memory usage: 8.1+ MB
```

In [22]:
```
# Calculate the percentage of missing values in each column
(df.isnull().sum() * 100 / len(df)).round(2)
```

Out[22]:
```
Investigation.Type        0.0
Event.Date                0.0
Injury.Severity           0.0
Aircraft.damage           0.0
Make                      0.0
Model                     0.0
Number.of.Engines         0.0
Engine.Type               0.0
Purpose.of.flight         0.0
Total.Fatal.Injuries      0.0
Total.Serious.Injuries    0.0
Total.Minor.Injuries      0.0
Total.Uninjured           0.0
Weather.Condition         0.0
Broad.phase.of.flight     0.0
Report.Status             0.0
Maker_Model               0.0
General_Maker_Model       0.0
Total_On_Board            0.0
dtype: float64
```

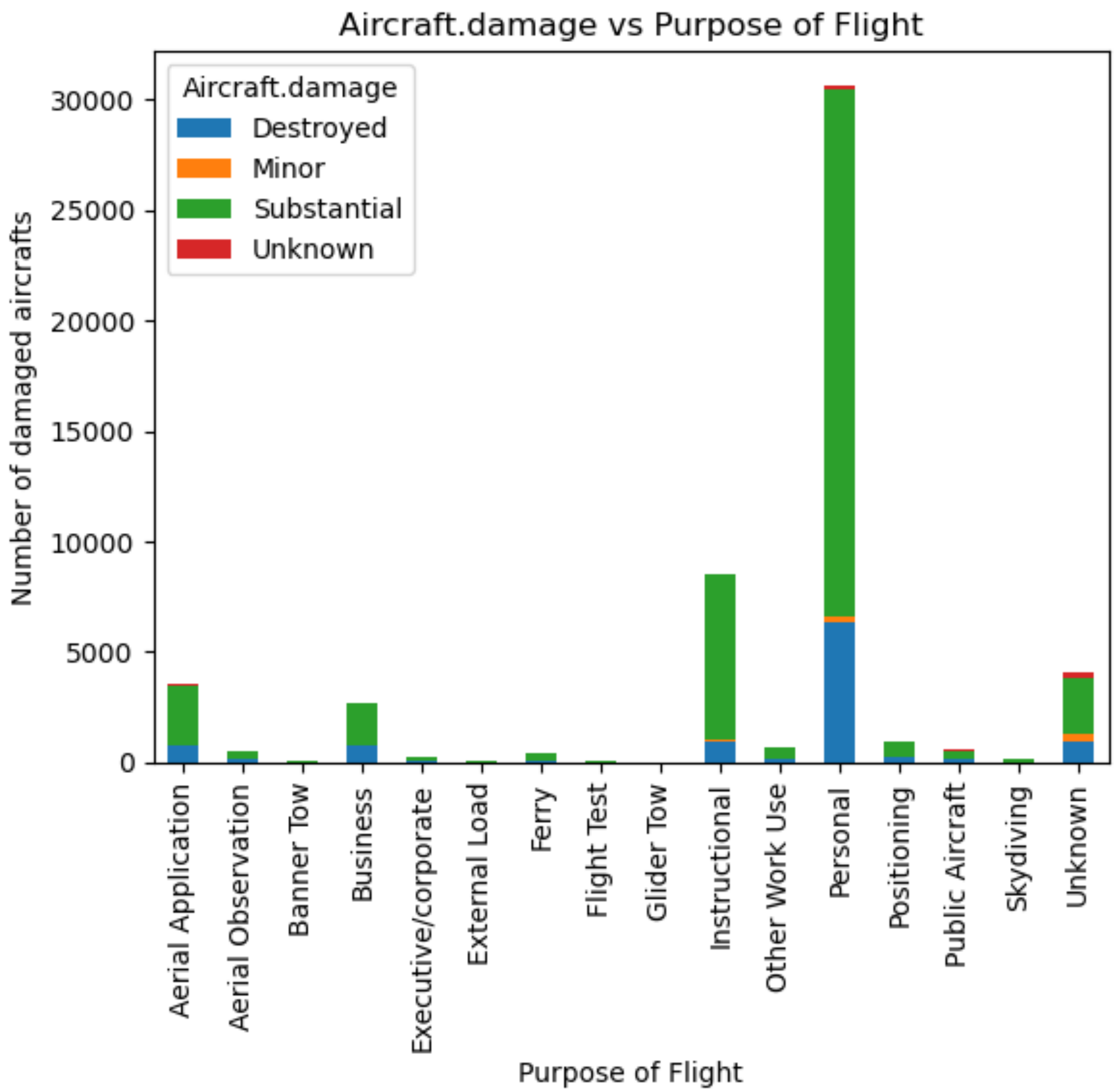
Let's plot 'Aircraft.Damage' attribute against others

Purpose of Flight

In [23]:
```
# Calculate value counts
purpose_value_counts = df['Purpose.of.flight'].value_counts()

# Filter the DataFrame
purpose_df = df[df['Purpose.of.flight'].isin(purpose_value_counts[purp
purpose_df.shape
```

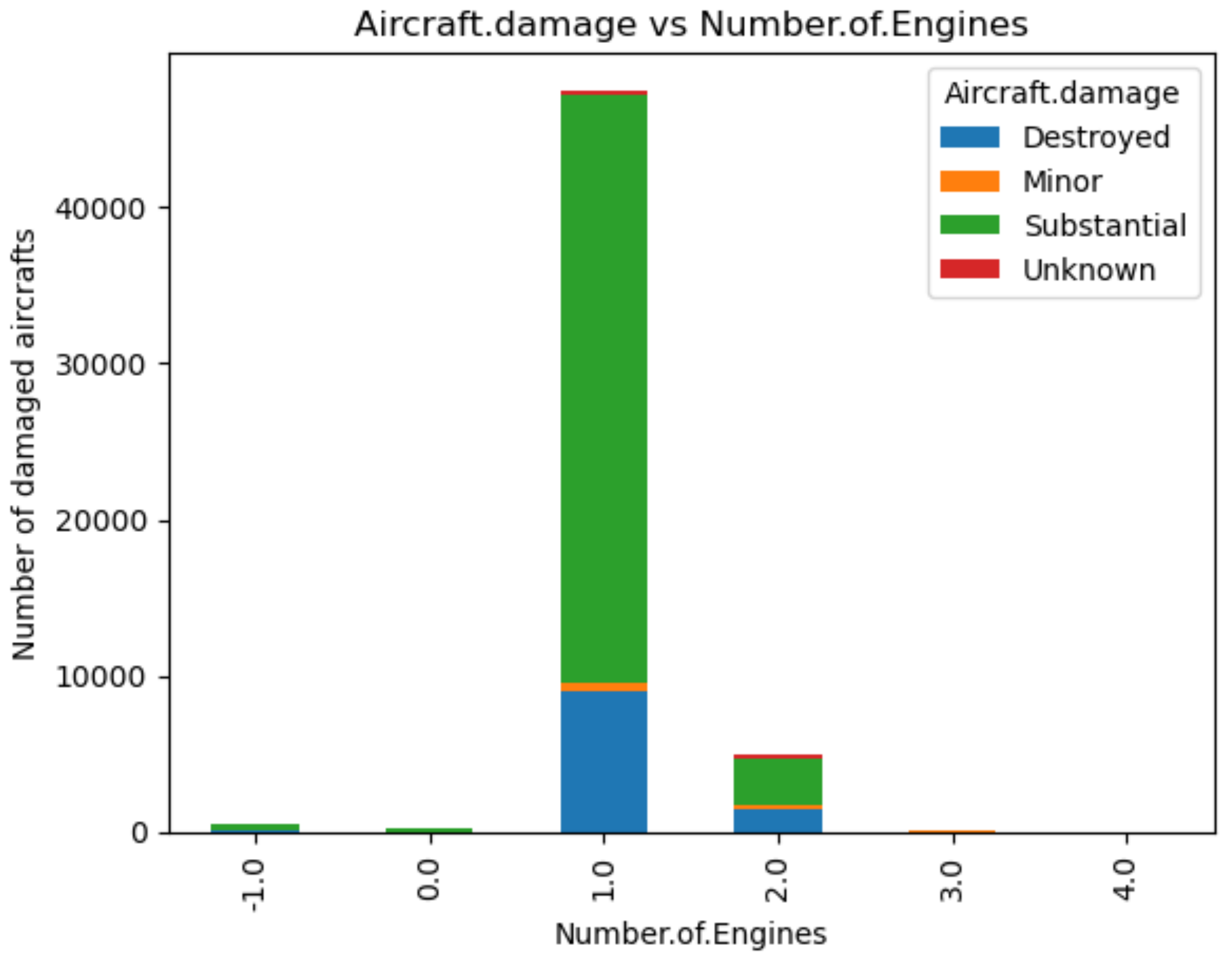
Out[23]:
```
(53233, 19)
```

In [24]:
```
# Create a bar plot
purpose_df.groupby(['Purpose.of.flight', 'Aircraft.damage']).size().ur
plt.xlabel('Purpose of Flight')
plt.ylabel('Number of damaged aircrafts')
plt.title('Aircraft.damage vs Purpose of Flight')
plt.show()
```



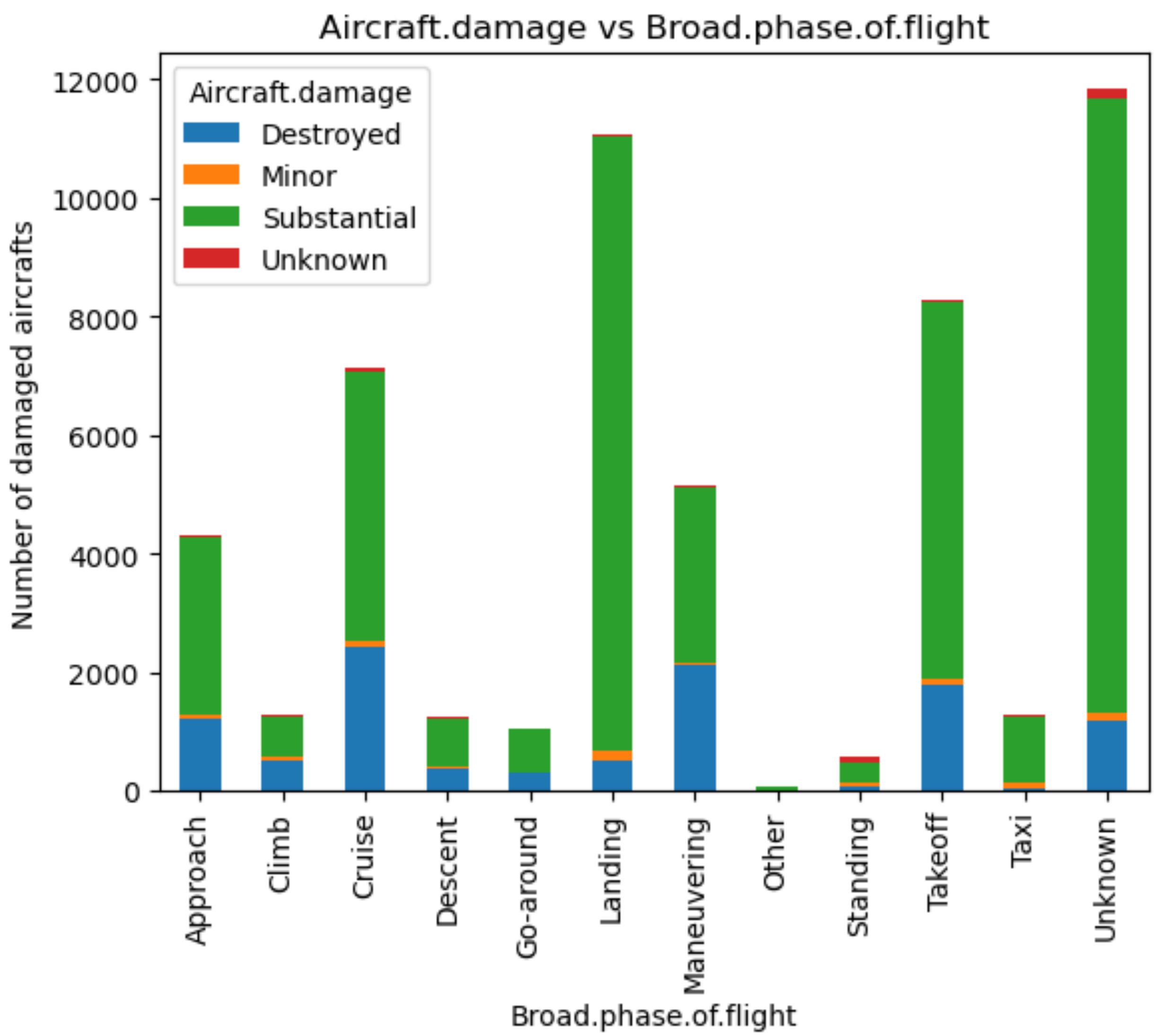
Number of Engines

In [25]:
```
df.groupby(['Number.of.Engines', 'Aircraft.damage']).size().unstack().
plt.xlabel('Number.of.Engines')
plt.ylabel('Number of damaged aircrafts')
plt.title('Aircraft.damage vs Number.of.Engines')
plt.show()
```



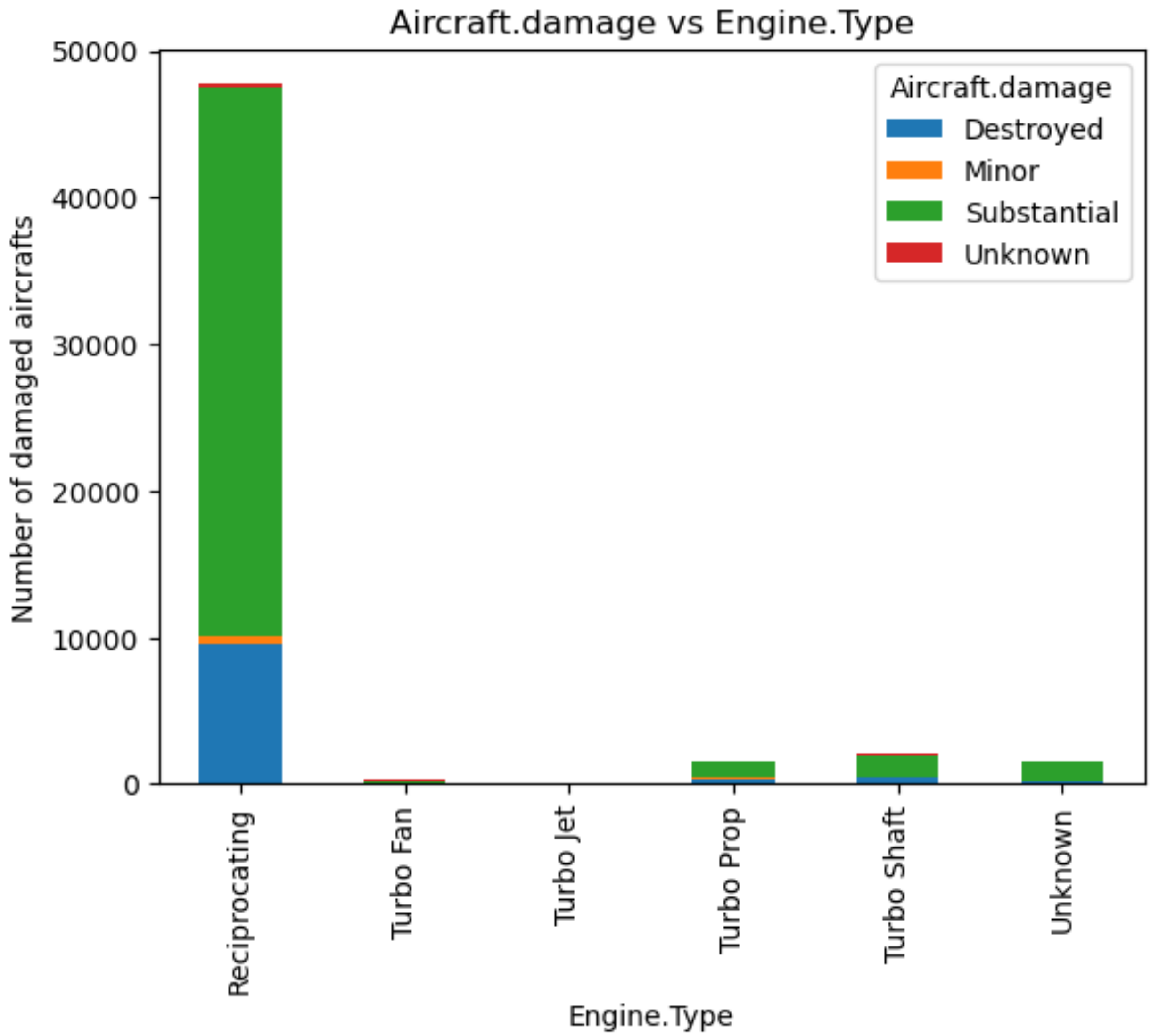
Broad Phase of Flight

In [26]:
```
df.groupby(['Broad.phase.of.flight', 'Aircraft.damage']).size().unstac
plt.xlabel('Broad.phase.of.flight')
plt.ylabel('Number of damaged aircrafts')
plt.title('Aircraft.damage vs Broad.phase.of.flight')
plt.show()
```



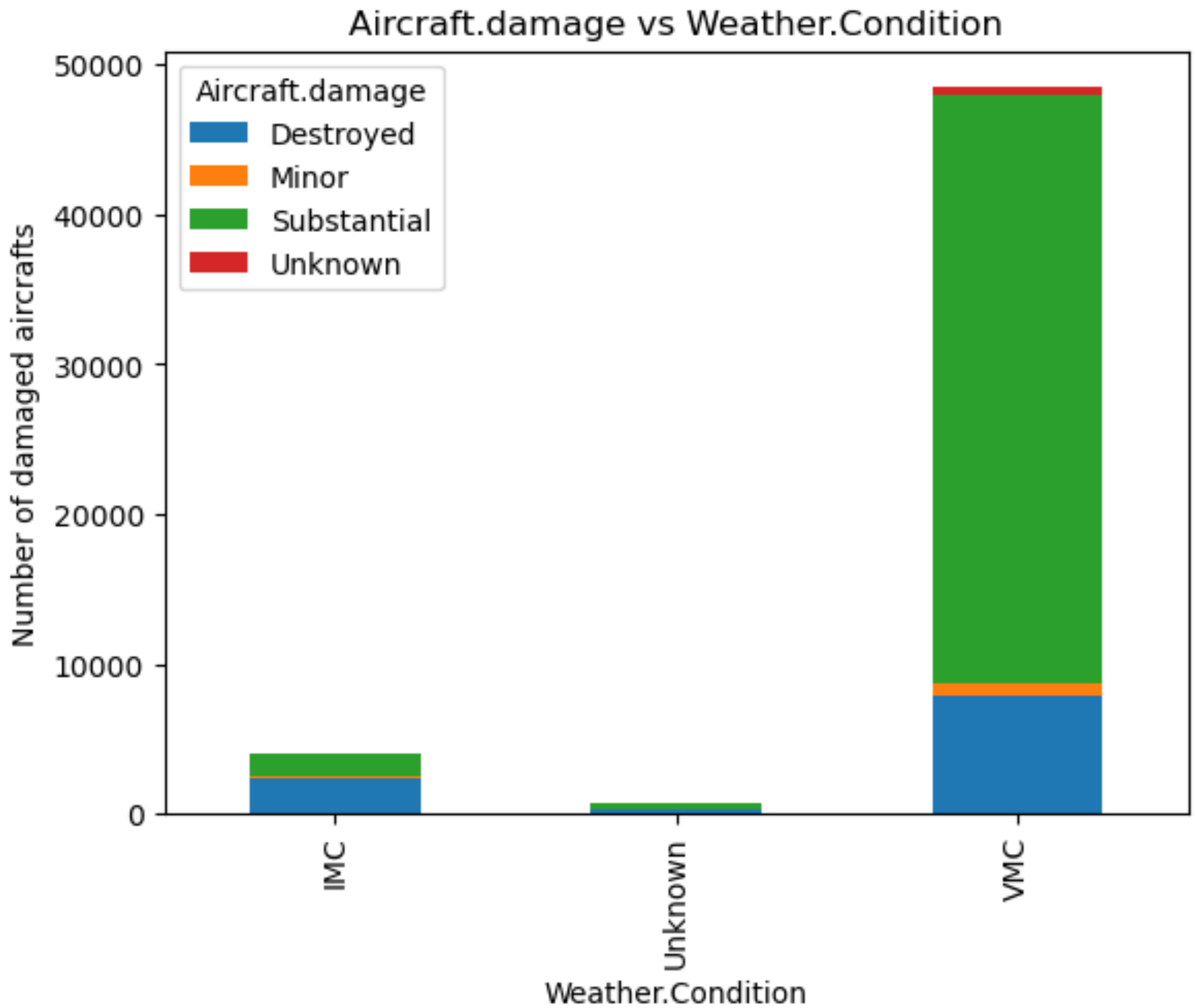
Engine Type

In [27]:
```
df.groupby(['Engine.Type', 'Aircraft.damage']).size().unstack().plot(k
plt.xlabel('Engine.Type')
plt.ylabel('Number of damaged aircrafts')
plt.title('Aircraft.damage vs Engine.Type')
plt.show()
```



Weather Condition

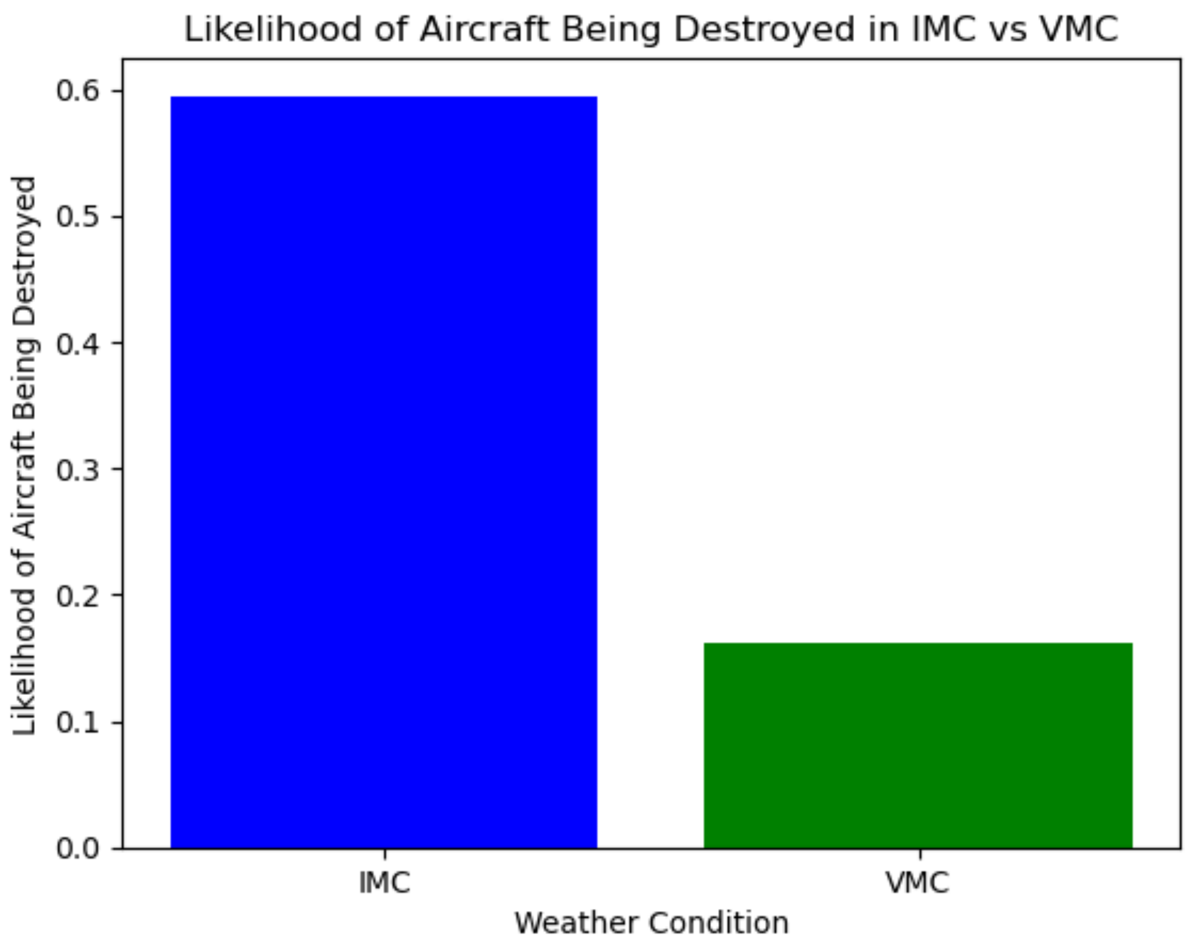
In [28]:
```
df.groupby(['Weather.Condition', 'Aircraft.damage']).size().unstack().
plt.xlabel('Weather.Condition')
plt.ylabel('Number of damaged aircrafts')
plt.title('Aircraft.damage vs Weather.Condition')
plt.show()
```



Let's calculate the likelyhood of the aircraft being destroyed in different weather conditions

In [29]:
```
# Filter the DataFrame for IMC and VMC conditions
imc_df = df[df['Weather.Condition'] == 'IMC']
vmc_df = df[df['Weather.Condition'] == 'VMC']

# Calculate the number of destroyed aircraft in each condition
destroyed_imc = imc_df[imc_df['Aircraft.damage'] == 'Destroyed'].shape
destroyed_vmc = vmc_df[vmc_df['Aircraft.damage'] == 'Destroyed'].shape

# Calculate the total number of aircraft in each condition
total_imc = imc_df.shape[0]
```

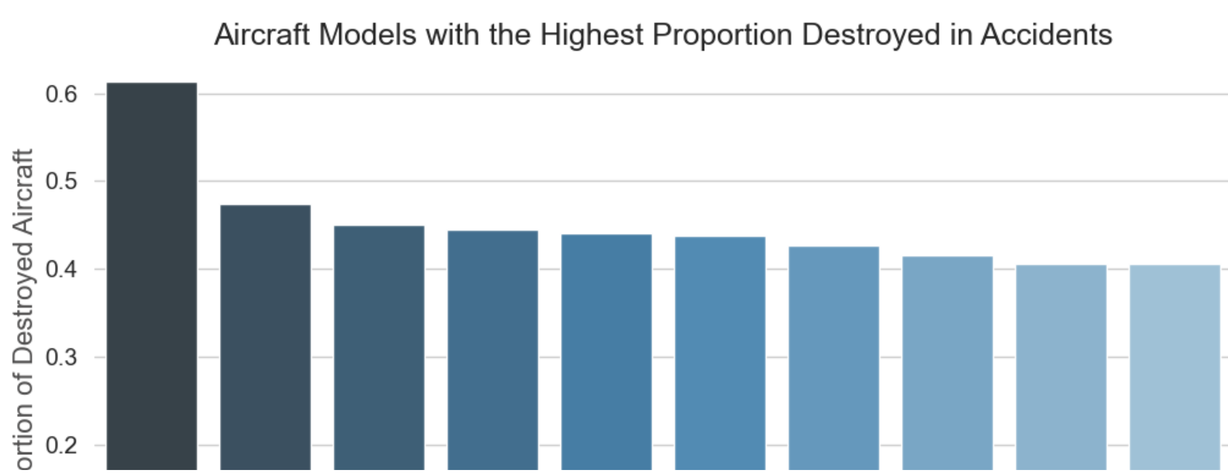
```
total_vmc = vmc_df.shape[0]

# Calculate the likelihood
likelihood_imc = destroyed_imc / total_imc if total_imc > 0 else 0
likelihood_vmc = destroyed_vmc / total_vmc if total_vmc > 0 else 0

# Plot the results
conditions = ['IMC', 'VMC']
likelihoods = [likelihood_imc, likelihood_vmc]

plt.bar(conditions, likelihoods, color=['blue', 'green'])
plt.xlabel('Weather Condition')
plt.ylabel('Likelihood of Aircraft Being Destroyed')
plt.title('Likelihood of Aircraft Being Destroyed in IMC vs VMC')
plt.show()
```

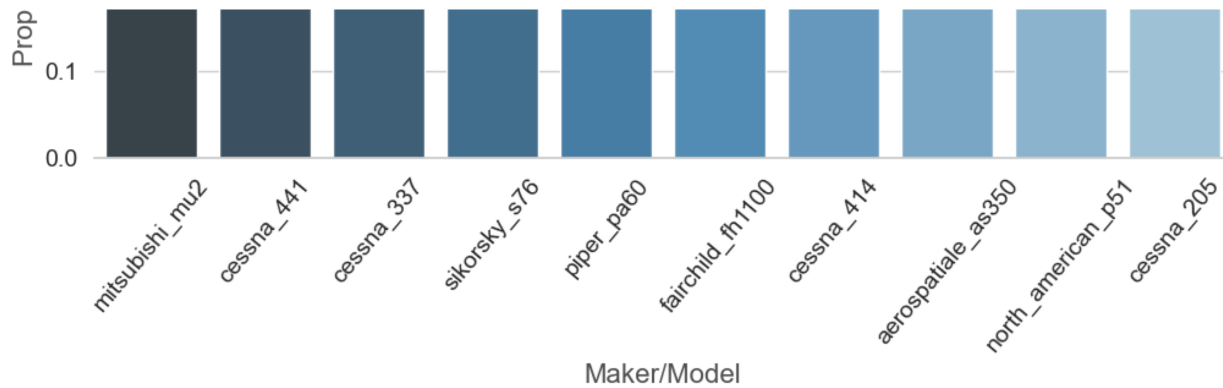


Let's get top 10 aircraft models with the highest proportion destroyed in accidents

In [31]:
```
# Calculate the percentage of 'Destroyed' aircraft for each General_Ma
```

```
damage_counts = df.groupby('General_Maker_Model')['Aircraft.damage'].v
#damage_counts['Destroyed_Percentage'] = damage_counts['Destroyed'] *
damage_counts['Destroyed_Percentage'] = damage_counts['Destroyed']

# Sort by Destroyed_Percentage and select the top 10
top_10_damage_counts = damage_counts.sort_values(by='Destroyed_Percent

# Reverse the color palette
reversed_palette = sns.color_palette('Blues_d', n_colors=10)[::-1]

# Plot the data using Seaborn
# Change the Size of the Chart
plt.figure(figsize=(12, 6))

# Increase the sharpness of the display
plt.rcParams['figure.dpi'] = 360

# add horizontal grid lines to the background
sns.set(style="whitegrid")

sns.barplot(x=top_10_damage_counts.index, y=top_10_damage_counts['Dest
plt.xlabel('Maker/Model', size=16, color='#4f4e4e')
# Change the color of y-axis and x-axis labels to dark grey
plt.ylabel('Proportion of Destroyed Aircraft', size=16, color='#4f4e4e
plt.title('Aircraft Models with the Highest Proportion Destroyed in Ac
plt.xticks(rotation=50)

# Make the Axis Tick Labels Bigger
plt.xticks(size=14)
plt.yticks(size=14)

# Remove Top and Right Border
#sns.despine()
# Remove Left Border
sns.despine(left=True)

plt.show()
```

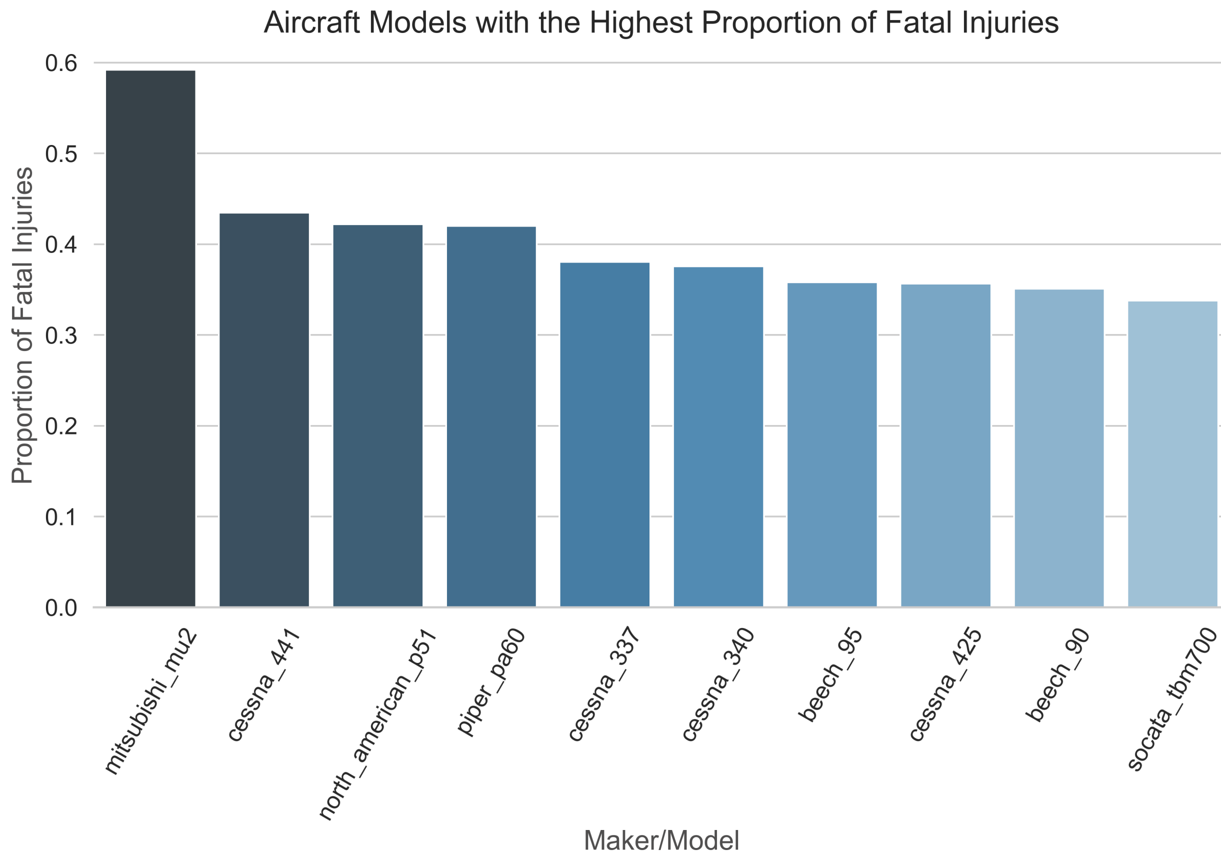




Let's get top 10 aircraft models with the highest proportion of fatal injuries

In [32]:
```
# Calculate the proportion of Total.Fatal.Injuries out of Total_On_Boa
df.loc[:, 'Proportion_Fatal'] = df['Total.Fatal.Injuries'] / df['Total

# Group by General_Maker_Model and calculate the mean proportion
grouped_df = df.groupby('General_Maker_Model')['Proportion_Fatal'].mea

# Sort by Proportion_Fatal and select the top 10
top_10_df = grouped_df.sort_values(by='Proportion_Fatal', ascending=Fa

# Reverse the color palette
reversed_palette = sns.color_palette('Blues_d', n_colors=10)[::-1]

# Plot the data using Seaborn
plt.figure(figsize=(12, 6))

# Increase the sharpness of the display
plt.rcParams['figure.dpi'] = 360

# Add horizontal grid lines to the background
sns.set(style="whitegrid")

# Use the correct hue parameter
sns.barplot(x='General_Maker_Model', y='Proportion_Fatal', data=top_10
plt.xlabel('Maker/Model', size=16, color='#4f4e4e')
plt.ylabel('Proportion of Fatal Injuries', size=16, color='#4f4e4e')
plt.title('Aircraft Models with the Highest Proportion of Fatal Injuri
plt.xticks(rotation=60)  # Rotate x-axis labels for better readability

# Make the Axis Tick Labels Bigger
plt.xticks(size=14)
plt.yticks(size=14)

# Remove Top and Right Border
```

```
sns.despine(left=True)

# Hide the legend
plt.legend([],[], frameon=False)

plt.show()
```



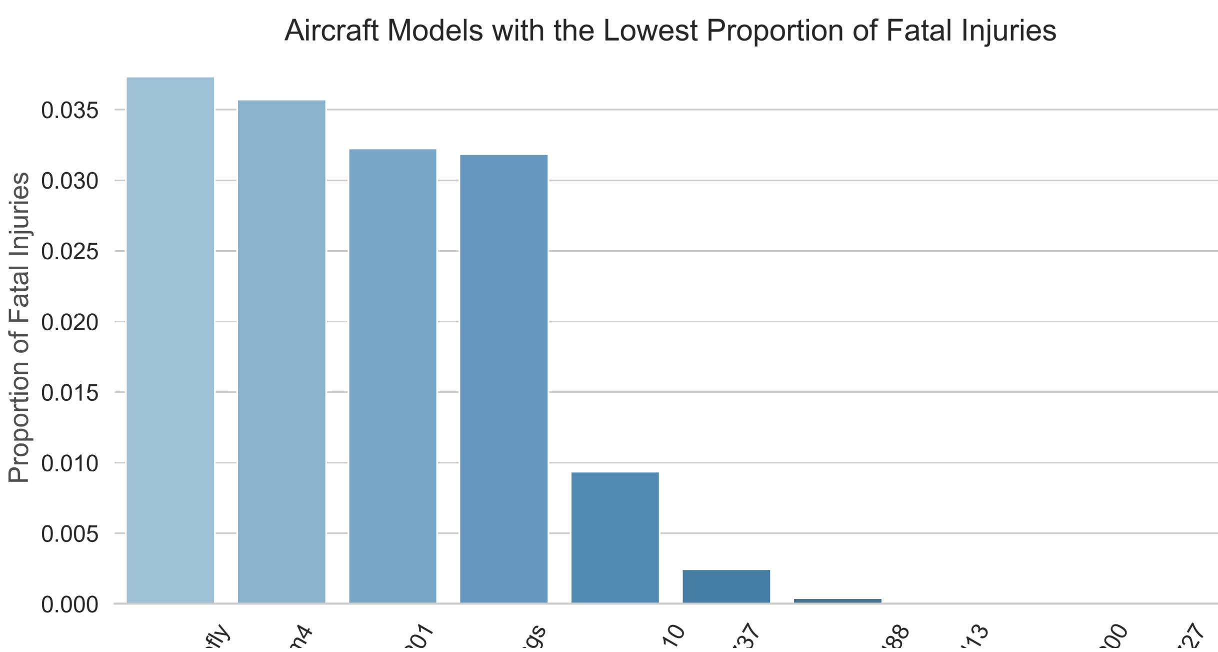
In [38]:
```
df['General_Maker_Model'].value_counts().get('mitsubishi_mu2', 0)
```

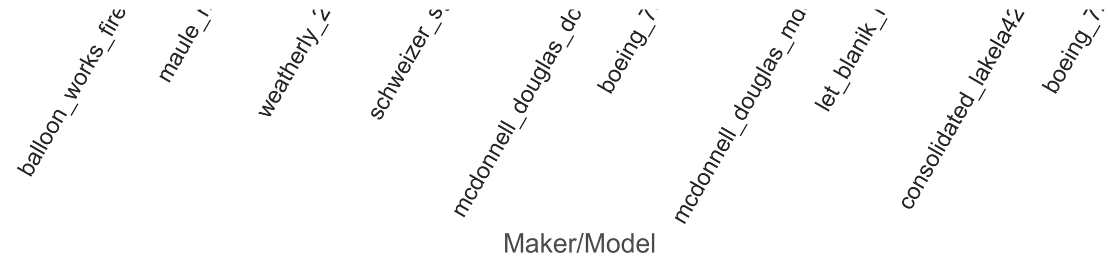
Out[38]: 31

Let's get top 10 aircraft models with the lowest proportion of fatal injuries

In [33]:
```
# Calculate the proportion of Total.Fatal.Injuries out of Total_On_Boa
df.loc[:, 'Proportion_Fatal'] = df['Total.Fatal.Injuries'] / df['Total

# Group by General_Maker_Model and calculate the mean proportion
grouped_df = df.groupby('General_Maker_Model')['Proportion_Fatal'].mea

# Sort by Proportion_Fatal and select the top 10
top_10_df = grouped_df.sort_values(by='Proportion_Fatal', ascending=Fa
```

```
# Use the original color palette
original_palette = sns.color_palette('Blues_d', n_colors=10)
#original_palette = sns.color_palette(['#000080', '#00008B', '#0000CD'


# Plot the data using Seaborn
plt.figure(figsize=(12, 6))

# Increase the sharpness of the display
plt.rcParams['figure.dpi'] = 360

# Add horizontal grid lines to the background
sns.set(style="whitegrid")

# Use the correct hue parameter
sns.barplot(x='General_Maker_Model', y='Proportion_Fatal', data=top_10
plt.xlabel('Maker/Model', size=16, color='#4f4e4e')
plt.ylabel('Proportion of Fatal Injuries', size=16, color='#4f4e4e')
plt.title('Aircraft Models with the Lowest Proportion of Fatal Injurie
plt.xticks(rotation=60)  # Rotate x-axis labels for better readability

# Make the Axis Tick Labels Bigger
plt.xticks(size=14)
plt.yticks(size=14)

# Remove Top and Right Border
sns.despine(left=True)

# Hide the legend
plt.legend([],[], frameon=False)

plt.show()
```





In [34]:
```
df[df['General_Maker_Model']=='balloon_works_firefly']
```

Out[34]:

Event.Id	Investigation.Type	Event.Date	Injury.Severity	Aircraft.damage	Make	Model
20020917X02927	Accident	1982-05-20	Non-Fatal	Substantial	balloon works	firefly7
20020917X02772	Accident	1982-07-05	Non-Fatal	Substantial	balloon works	firefly7
20020917X04680	Accident	1982-10-02	Non-Fatal	Unknown	balloon works	firefly7
20020917X04676	Accident	1982-10-04	Non-Fatal	Substantial	balloon works	firefly7
20001214X42945	Accident	1983-05-30	Non-Fatal	Unknown	balloon works	firefly7
20001214X43009	Accident	1983-05-30	Non-Fatal	Substantial	balloon works	firefly7
20001214X43265	Accident	1983-06-	Fatal(1)	Substantial	balloon	firefly7

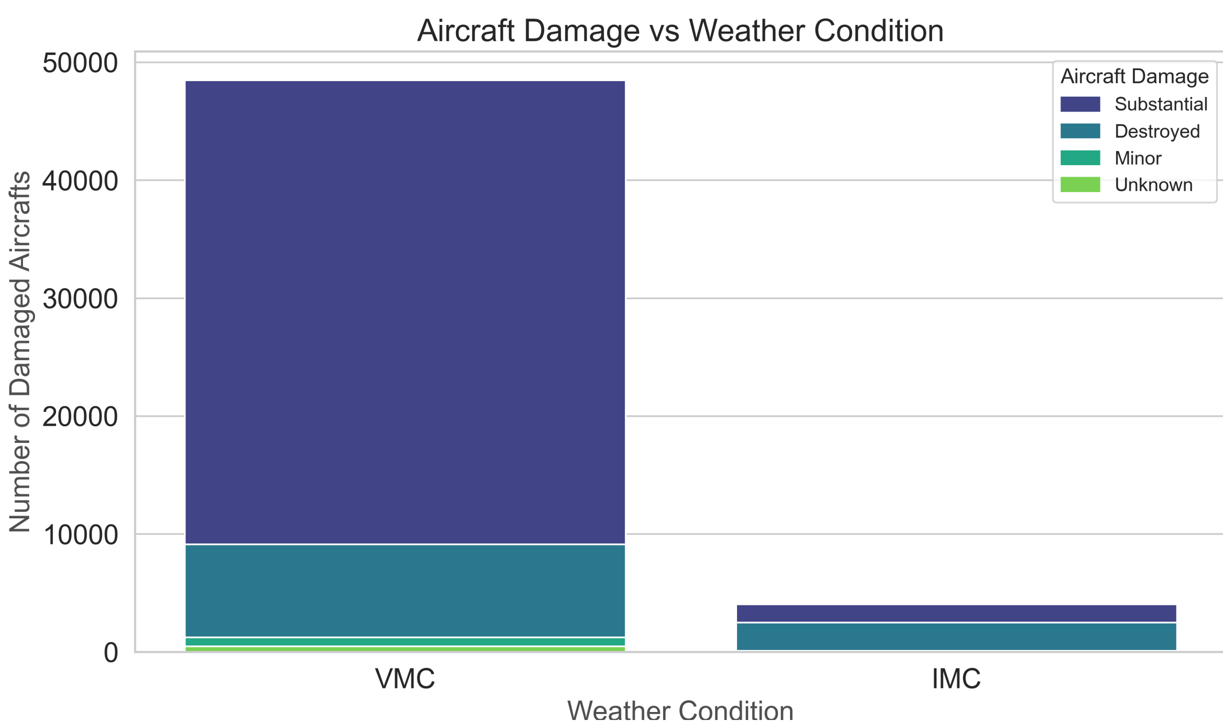
In [35]:
```
df['General_Maker_Model'].value_counts().get('balloon_works_firefly',
```

Out[35]: 50

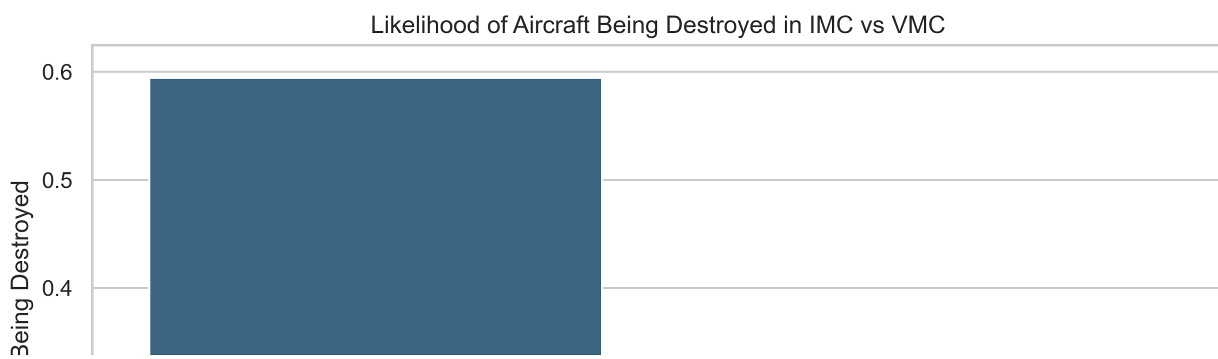
Are risks higher in poor weather?

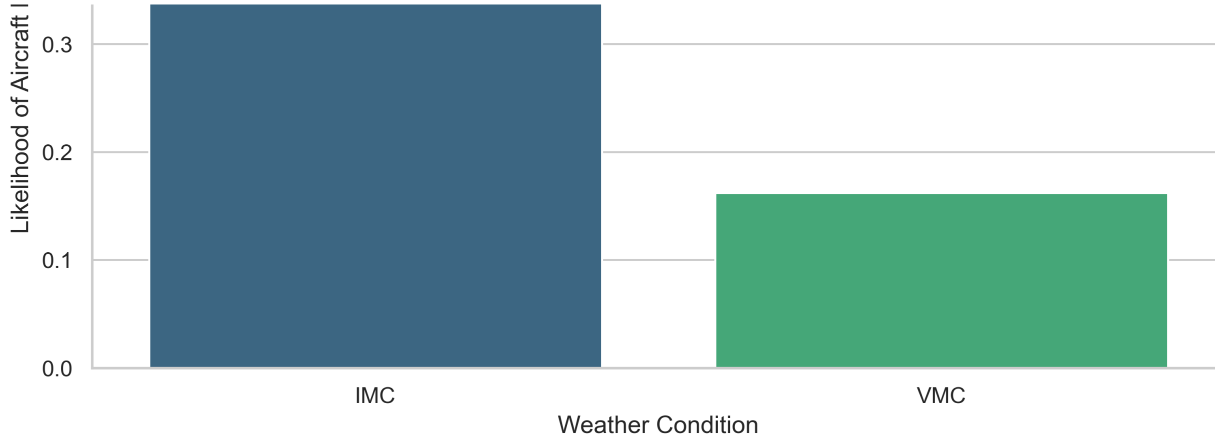
In [36]:
```
# Filter out rows where Weather.Condition is unknown
filtered_df = df[df['Weather.Condition'] != 'Unknown']

# Create the plot
plt.figure(figsize=(10, 6))
plot = sns.histplot(data=filtered_df, x='Weather.Condition', hue='Airc

# Increase the sharpness of the display
plt.rcParams['figure.dpi'] = 360

# Customize the plot
```

```
plot.set_facecolor('white')
plot.grid(axis='x', visible=False)  # Remove vertical grid lines
plot.grid(axis='y', visible=True)   # Keep horizontal grid lines

# Manually create legend with actual colors
color_palette = sns.color_palette('viridis', n_colors=4)
color_explanations = {
    'Substantial': color_palette[0],
    'Destroyed': color_palette[1],
    'Minor': color_palette[2],
    'Unknown': color_palette[3]
}

# Create custom legend patches
legend_patches = [mpatches.Patch(color=color, label=label) for label,

plt.legend(handles=legend_patches, title='Aircraft Damage', loc='upper

plt.xlabel('Weather Condition', size=16, color='#4f4e4e')
plt.ylabel('Number of Damaged Aircrafts', size=16, color='#4f4e4e')
plt.title('Aircraft Damage vs Weather Condition', size=18)
plt.xticks(rotation=0)
plt.tight_layout()

# Make the Axis Tick Labels Bigger
plt.xticks(size=16)
plt.yticks(size=16)


plt.show()
```



In [37]:
```
# Filter the DataFrame for IMC and VMC conditions
imc_df = df[df['Weather.Condition'] == 'IMC']
vmc_df = df[df['Weather.Condition'] == 'VMC']

# Calculate the number of destroyed aircraft in each condition
destroyed_imc = imc_df[imc_df['Aircraft.damage'] == 'Destroyed'].shape
destroyed_vmc = vmc_df[vmc_df['Aircraft.damage'] == 'Destroyed'].shape

# Calculate the total number of aircraft in each condition
total_imc = imc_df.shape[0]
total_vmc = vmc_df.shape[0]

# Calculate the likelihood
likelihood_imc = destroyed_imc / total_imc if total_imc > 0 else 0
likelihood_vmc = destroyed_vmc / total_vmc if total_vmc > 0 else 0

# Prepare data for plotting
data = {
    'Weather Condition': ['IMC', 'VMC'],
    'Likelihood of Aircraft Being Destroyed': [likelihood_imc, likelih
}

# Create a DataFrame
plot_df = pd.DataFrame(data)

# Define the color palette
color_palette = sns.color_palette('viridis', n_colors=2)

# Create the plot
plt.figure(figsize=(10, 6))
sns.barplot(x='Weather Condition', y='Likelihood of Aircraft Being Des

# Customize the plot
plt.xlabel('Weather Condition')
plt.ylabel('Likelihood of Aircraft Being Destroyed')
plt.title('Likelihood of Aircraft Being Destroyed in IMC vs VMC')
plt.show()
```





Based on the above findings, we can make the following recommendations:

- Avoid purchasing aircraft models with high fatality and destruction rates;
- Invest in pilot training and maintenance for operations in IMC to mitigate risk.

In [ ]: