mbarriola96 /
Aircraft_safety_analysis

Code Issues Pull requests Actions Projects Wiki Security

mbarriola96 More final updated 7 minutes ago

5757 lines (5757 loc) · 1.38 MB

Preview Code Blame Raw

1. Overview

This project analyzes aircraft accidents and incidents from 1948 to 2022. It is part of the initial phase of the FlatIron Data Science bootcamp, with the requirement to investigate the provided dataset. The goal is to derive three business recommendations for strategic investments in the aircraft industry. The business problem is to identify low-risk aircraft for a company looking to expand into commercial and private aviation.

2. Business Understanding

The main objective is to discern which aircraft present the lowest risk for the company's venture into aviation. This expansion requires a thorough risk assessment to make informed decisions on aircraft acquisition. The findings will be translated into actionable insights for the head of the new aviation division to guide purchase decisions. The investigation centers on assessing the risk profiles of various aircraft, with the aim of providing three informed business recommendations. These recommendations will specifically address which types of aircraft the company should consider for investment based on historical safety data. The ultimate goal is to guide the company towards aircraft options that minimize risk and potential liability, thereby supporting safe and sound investment decisions in the aviation sector.

Our primary stakeholders are the board members of the company as they are the ones to decide whether to carry out the investment or not.

3. Data Understanding

3.1 Data Description

For the project, the data source is drawn from Kaggle, which encompasses a comprehensive collection of aircraft accidents and incidents. The timeline of this dataset spans an extensive period, covering events from the year 1948 through to 2022.

The dataset has undergone a meticulous cleaning procedure to ensure the quality and relevance of the data. This process included a filter to retain only those incidents and accidents that occurred within the United States. Additionally, the data was refined by filtering out events to include only those that resulted in fatal injuries or serious injuries, thus focusing on the most severe occurrences.

Now let's dive into the data to better understand it and arrive to the business recommendations.

3.2 SetUp

3.3 Import necessary libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:

```
pd.set_option('display.max_columns', 500)
```

3.4 Define global variables

In [3]:

```
INPUT_PATH = "../../Data_Project_Phase1/AviationData.csv"
```

In [4]:

```
!pwd
```

```
/c/Users/Usuario/Desktop/FlatIron/DataScience_FlatIron_Curso/Phase_1/Phase1-Pr
oject/Aircraft_safety_analysis/notebooks
```

3.5 Functions

In [5]:

```
def categorize_data(column):
    """
    Function: This function will return the string 'zero' if the value of the
    'one or more' if the value of 'column' is not zero

    Argument (data series): The column to evaluate

    Result (string): The category label for the value

    """
    if column==0:
        return 'Zero'
    elif pd.isna(column):
        return 'Unknown'
    else:
        return 'One or More'
```

In [6]:

```
def plot_bar_graph_for_columns(columns):

    """
    Function: This function creates a bar graph for a column.

    Argument (data series): The columns to evaluate.

    Returns: Bar plot for the column

    """

    plt.figure()
    df[columns].value_counts().plot(kind='bar')
    plt.xlabel(columns)
    plt.ylabel('Frequency')
    plt.xticks(rotation=90)
    plt.title(f'Bar Graph of {columns}')
```

In [7]:

```
def plot_column_data(df, column, kind_of_graph):

    """
    Function: This function creates a value_counts and the desired graph for

    Argument (data series): The data frame, the coolumn and the kind of graph

    Returns: Value_counts of the column and the desired graph representation
    """

    # Print the normalized value counts including NaN values
    value_counts = df[column].value_counts(normalize=True, dropna=False)
    print(value_counts)
    print()

    # Plot the graph
    if kind_of_graph == 'bar':
        plt.figure()
        value_counts.plot(kind='bar')
    elif kind_of_graph == 'pie':
        plt.figure()
        value_counts.plot(kind='pie')
    elif kind_of_graph == 'line':
        plt.figure()
        value_counts.plot(kind='line')

    # Show the plot
    plt.title(f'Graph of {column}')
    plt.ylabel('Frequency')
    plt.xlabel(column)
    plt.xticks(rotation=90)
    plt.show();
```

In [8]:

```
def plot_feature(df: pd.DataFrame,
                 column_name: str,
```

```
                column_name: str,
                column_type: str,
                variable_target1: str,
                variable_target2: str):
    """
    Visualize a variable with faceting on two target variables.

    Parameters:
        df (pd.DataFrame): The dataframe containing the data.
        column_name (str): The name of the column to be visualized.
        column_type (str): The type of the column ('continuous' or 'categoric
        variable_target1 (str): The name of the first target variable for fac
        variable_target2 (str): The name of the second target variable for fa
    """
    f, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(18,6), dpi=9

    # Plot without target variables
    if column_type == 'continuous':
        sns.distplot(df.loc[df[column_name].notnull(), column_name], kde=Fals
    else:
        categories_to_consider = list(df[column_name].value_counts().index[:1
        df = df[df[column_name].isin(categories_to_consider)]
        sns.countplot(x=df[column_name], order=sorted(categories_to_consider)
                      color='#5975A4', saturation=1, ax=ax1)
    ax1.set_xlabel(column_name)
    ax1.set_ylabel('Count')
    ax1.set_title(f"Distribution of {column_name}")
    ax1.tick_params(axis='x', rotation=90)

    # Plot with the first target variable
    if column_type == "continuous":
        sns.boxplot(x=column_name, y=variable_target1, data=df, ax=ax2)
    else:
        data = df.groupby(column_name)[variable_target1].value_counts(normali
        data.plot(kind='bar', stacked=True, ax=ax2)
    ax2.set_ylabel(f"Proportion of {variable_target1}")
    ax2.set_title(f"{column_name} by {variable_target1}")
    ax2.tick_params(axis='x', rotation=90)

    # Plot with the second target variable
    if column_type == "continuous":
        sns.boxplot(x=column_name, y=variable_target2, data=df, ax=ax3)
    else:
        data = df.groupby(column_name)[variable_target2].value_counts(normali
        data.plot(kind='bar', stacked=True, ax=ax3)
    ax3.set_ylabel(f"Proportion of {variable_target2}")
    ax3.set_title(f"{column_name} by {variable_target2}")
    ax3.tick_params(axis='x', rotation=90)

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

3.6 Code

In [9]:

```
df = pd.read_csv(INPUT_PATH, encoding="latin-1")
df
```

```
C:\Users\Usuario\AppData\Local\Temp\ipykernel_17664\281516245.py:1: DtypeWarni
ng: Columns (6,7,28) have mixed types. Specify dtype option on import or set l
ow_memory=False.
  df = pd.read_csv(INPUT_PATH, encoding="latin-1")
```

Out[9]:

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Locati
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MOO CREEK,
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPOI (
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, \
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, (
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, C
...	...	...	...	...	
88884	20221227106491	Accident	ERA23LA093	2022-12-26	Annapol M
88885	20221227106494	Accident	ERA23LA095	2022-12-26	Hamptc N
88886	20221227106497	Accident	WPR23LA075	2022-12-26	Payson, /
88887	20221227106498	Accident	WPR23LA076	2022-12-26	Morgan, l
88888	20221230106513	Accident	ERA23LA097	2022-12-29	Athens, (

88889 rows × 31 columns

In [10]:

```
print(f"This dataset has {df.shape[0]} rows and {df.shape[1]} columns")
```

```
This dataset has 88889 rows and 31 columns
```

In [11]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 88889 entries, 0 to 88888
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Event.Id                88889 non-null  object
 1   Investigation.Type      88889 non-null  object
 2   Accident.Number         88889 non-null  object
```

```
 2   Accident.Number         88889 non-null  object
 3   Event.Date              88889 non-null  object
 4   Location                88837 non-null  object
 5   Country                 88663 non-null  object
 6   Latitude                34382 non-null  object
 7   Longitude               34373 non-null  object
 8   Airport.Code            50132 non-null  object
 9   Airport.Name            52704 non-null  object
 10  Injury.Severity         87889 non-null  object
 11  Aircraft.damage         85695 non-null  object
 12  Aircraft.Category       32287 non-null  object
 13  Registration.Number     87507 non-null  object
 14  Make                    88826 non-null  object
 15  Model                   88797 non-null  object
 16  Amateur.Built           88787 non-null  object
 17  Number.of.Engines       82805 non-null  float64
 18  Engine.Type             81793 non-null  object
 19  FAR.Description         32023 non-null  object
 20  Schedule                12582 non-null  object
 21  Purpose.of.flight       82697 non-null  object
 22  Air.carrier             16648 non-null  object
 23  Total.Fatal.Injuries    77488 non-null  float64
 24  Total.Serious.Injuries  76379 non-null  float64
 25  Total.Minor.Injuries    76956 non-null  float64
 26  Total.Uninjured         82977 non-null  float64
 27  Weather.Condition       84397 non-null  object
 28  Broad.phase.of.flight   61724 non-null  object
 29  Report.Status           82505 non-null  object
 30  Publication.Date        75118 non-null  object
dtypes: float64(5), object(26)
memory usage: 21.0+ MB
```

Now, I am going to clean the column names by making them be in lower case and using an underscore

In [12]:

```python
df.columns = df.columns.str.lower().str.replace('.', '_')
df.columns
```

Out[12]:

```
Index(['event_id', 'investigation_type', 'accident_number', 'event_date',
       'location', 'country', 'latitude', 'longitude', 'airport_code',
       'airport_name', 'injury_severity', 'aircraft_damage',
       'aircraft_category', 'registration_number', 'make', 'model',
       'amateur_built', 'number_of_engines', 'engine_type', 'far_description
',
       'schedule', 'purpose_of_flight', 'air_carrier', 'total_fatal_injuries
',
       'total_serious_injuries', 'total_minor_injuries', 'total_uninjured',
       'weather_condition', 'broad_phase_of_flight', 'report_status',
       'publication_date'],
      dtype='object')
```

In [13]:

```python
df
```

Out[13]:

	event_id	investigation_type	accident_number	event_date	locatic
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MOOS CREEK, I

1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPOR C
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, V
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, C
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, O
...	...	...	...	...	
88884	20221227106491	Accident	ERA23LA093	2022-12-26	Annapol M
88885	20221227106494	Accident	ERA23LA095	2022-12-26	Hampto N
88886	20221227106497	Accident	WPR23LA075	2022-12-26	Payson, A
88887	20221227106498	Accident	WPR23LA076	2022-12-26	Morgan, U
88888	20221230106513	Accident	ERA23LA097	2022-12-29	Athens, G

88889 rows × 31 columns

3.6.1 Descriptive Statistics

In [14]:

```
df.describe()
```

Out[14]:

	number_of_engines	total_fatal_injuries	total_serious_injuries	total_minor_injuri
count	82805.000000	77488.000000	76379.000000	76956.0000
mean	1.146585	0.647855	0.279881	0.3570
std	0.446510	5.485960	1.544084	2.2356
min	0.000000	0.000000	0.000000	0.0000
25%	1.000000	0.000000	0.000000	0.0000
50%	1.000000	0.000000	0.000000	0.0000
75%	1.000000	0.000000	0.000000	0.0000
max	8.000000	349.000000	161.000000	380.0000

Even though Number of Engines is continuous, it could be considered as descrete because it doesn't make much sense to talk about a mean of 1.14 of number of engines.

Other noticeable things are that there is a mean of almost 1 total fatal injury for all the accidents, and data seems to be coherent because there aren't negative values.

3.6.2 Making a primary key

In [15]:

```
df['event_id'].value_counts()
```

Out[15]:

```
event_id
20001212X19172    3
20001214X45071    3
20220730105623    2
20051213X01965    2
20001212X16765    2
                 ..
20001211X14216    1
20001211X14239    1
20001211X14207    1
20001211X14204    1
20221230106513    1
Name: count, Length: 87951, dtype: int64
```

In [16]:

```
df['accident_number'].value_counts()
```

Out[16]:

```
accident_number
CEN22LA149    2
WPR23LA041    2
WPR23LA045    2
DCA22WA214    2
DCA22WA089    2
             ..
LAX92FA065    1
ANC92T#A12    1
MIA92LA049    1
NYC92LA048    1
ERA23LA097    1
Name: count, Length: 88863, dtype: int64
```

In [17]:

```
df['registration_number'].value_counts()
```

Out[17]:

```
registration_number
NONE      344
UNREG     126
UNK        13
USAF        9
N20752      8
         ...
N93478      1
N519UA      1
N8840W      1
```

```
N21040      1
N9026P      1
Name: count, Length: 79104, dtype: int64
```

In [18]:

```
df[df['accident_number']=='CEN22LA149']
```

Out[18]:

	event_id	investigation_type	accident_number	event_date	location
87548	20220323104818	Accident	CEN22LA149	2022-03-18	Grapevine, TX
87549	20220323104818	Accident	CEN22LA149	2022-03-18	Grapevine, TX

In [19]:

```
df['primary_key'] = df['accident_number'] + '_' + df['registration_number']
df
```

Out[19]:

	event_id	investigation_type	accident_number	event_date	locatic
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MOOS CREEK, I
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPOR C
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, V
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, C
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, O
...	...	...	...	...	
88884	20221227106491	Accident	ERA23LA093	2022-12-26	Annapol M
88885	20221227106494	Accident	ERA23LA095	2022-12-26	Hampto N
88886	20221227106497	Accident	WPR23LA075	2022-12-26	Payson, A
88887	20221227106498	Accident	WPR23LA076	2022-12-26	Morgan, L
88888	20221230106513	Accident	ERA23LA097	2022-12-29	Athens, G

88889 rows × 32 columns

In [20]:
```python
df['primary_key'].value_counts()
```

Out[20]:
```
primary_key
SEA87LA080_NC6404     1
SEA05CA166_N2094K     1
CHI05CA172_N7446      1
DEN05CA122_N2584B     1
DEN05LA121_N5754S     1
                     ..
MIA91LA225_N2983U     1
ATL91LA180_N62108     1
ATL91LA181A_N26004    1
ATL91LA181B_N67174    1
ERA23LA097_N9026P     1
Name: count, Length: 87507, dtype: int64
```

We haven't found a primary key, but I have created one by combining 2 columns: accident_number and registration_number

3.6.3 Duplicates study

Checking for duplicates

In [21]:
```python
df.duplicated().sum()
```

Out[21]:
```
0
```

3.6.4 Null-values analysis

Checking for null values

In [22]:
```python
df.isnull().sum()/len(df)*100
```

Out[22]:
```
event_id                0.000000
investigation_type      0.000000
accident_number         0.000000
event_date              0.000000
location                0.058500
country                 0.254250
latitude               61.320298
longitude              61.330423
airport_code           43.601570
airport_name           40.708074
injury_severity         1.124999
aircraft_damage         3.593246
aircraft_category      63.677170
registration_number     1.554748
make                    0.070875
```

```
model                      0.103500
amateur_built              0.114750
number_of_engines          6.844491
engine_type                7.982990
far_description           63.974170
schedule                  85.845268
purpose_of_flight          6.965991
air_carrier               81.271023
total_fatal_injuries      12.826109
total_serious_injuries    14.073732
total_minor_injuries      13.424608
total_uninjured            6.650992
weather_condition          5.053494
broad_phase_of_flight     30.560587
report_status              7.181991
publication_date          15.492356
primary_key                1.554748
dtype: float64
```

I will proceed to create a list to drop certain columns that have too many null values and that I perceive not to be usefull for the analysis.

Latitude, Longitude, airpot_code, airport_name, and publication_date I decide to drop mainly because they are not useful for the case study. Schedule and air_carrier I decide to drop because they have more than 80% of null values

In [23]:

```
drop_columns = ['latitude','longitude','airport_code','airport_name','schedul
```

Now I will study other columns that have a high percent of null values to determine whether they still can give good insights. These columns are: aircraft_category, far_description, and broad_phase_of_flight

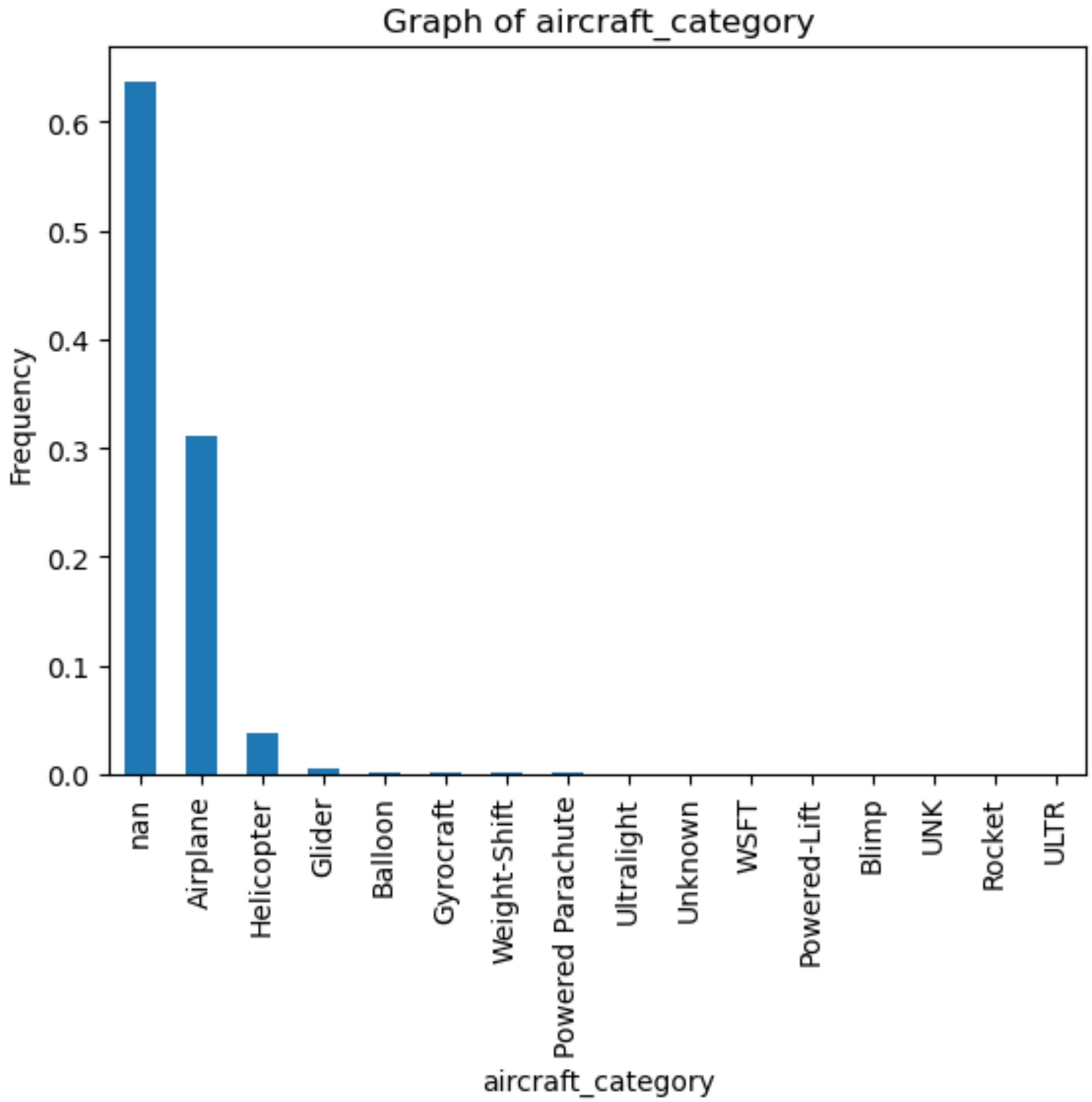
Aircraft_category

In [24]:

```
plot_column_data(df,'aircraft_category', 'bar')
```

```
aircraft_category
NaN                  0.636772
Airplane             0.310691
Helicopter           0.038700
Glider               0.005715
Balloon              0.002599
Gyrocraft            0.001946
Weight-Shift         0.001811
Powered Parachute    0.001024
Ultralight           0.000337
Unknown              0.000157
WSFT                 0.000101
Powered-Lift         0.000056
Blimp                0.000045
UNK                  0.000022
Rocket               0.000011
ULTR                 0.000011
Name: proportion, dtype: float64
```

```
Name: proportion, dtype: float64
```



It's observable that only the airplanes and the helicopters have considerable numbers of registrations in the aircraft category. Moreover, as can be seen most of the aircrafts are airplanes. Given that there 64% of NaN values, I will drop this column too

In [25]:
```
drop_columns.append('aircraft_category')
```

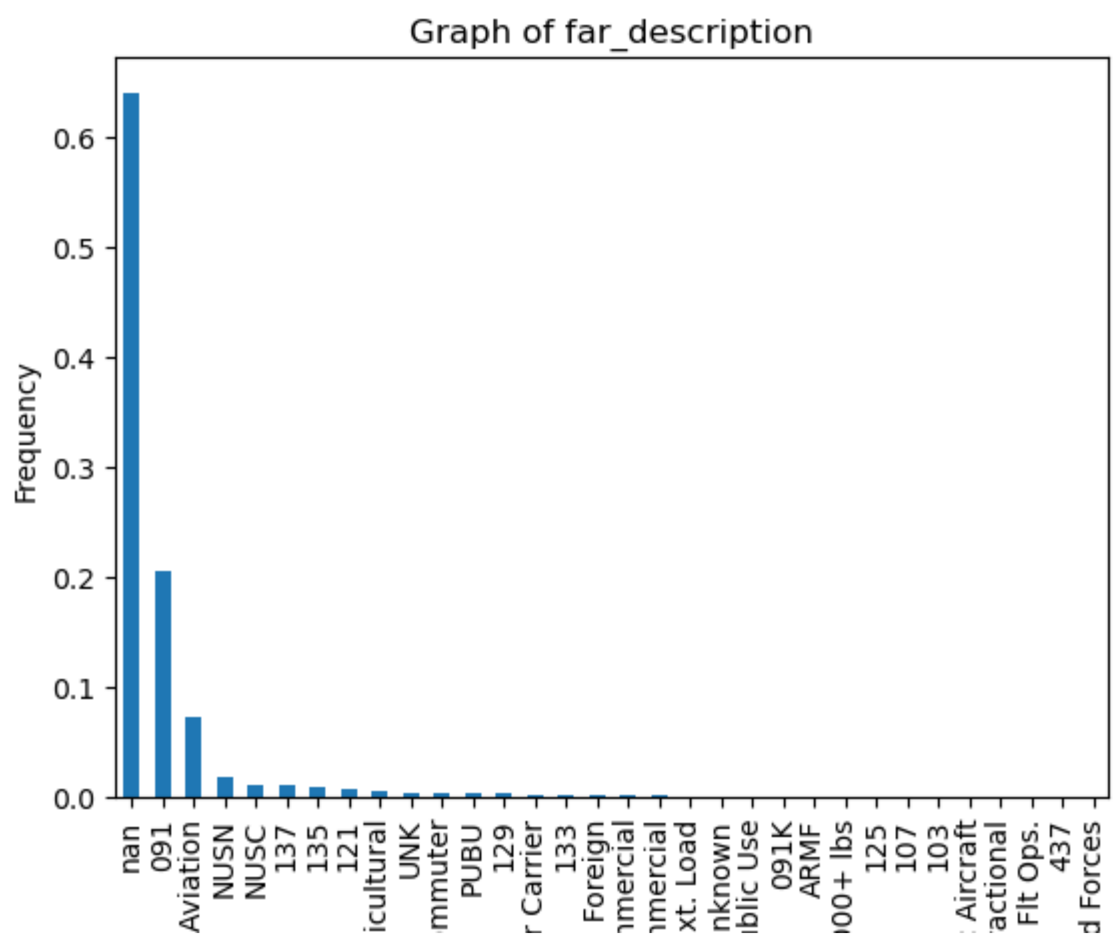
3.6.5 Further study of the rest of the columns

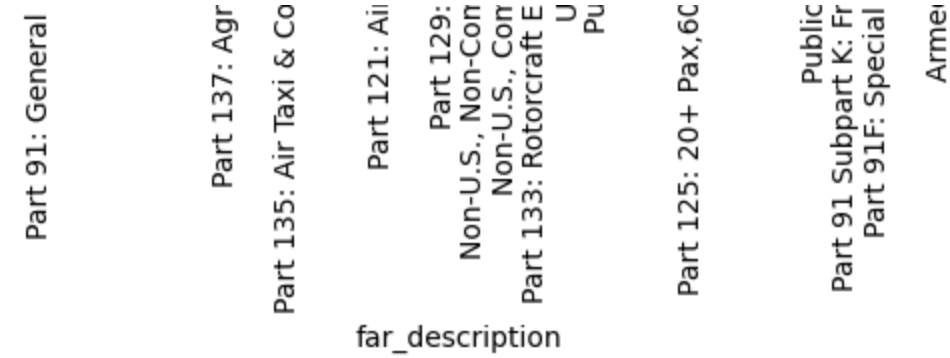
Far Description column

In [26]:
```
plot_column_data(df,'far_description', 'bar')
```

```
far_description
NaN                          0.639742
091                          0.204986
Part 91: General Aviation    0.072967
NUSN                         0.017820
```

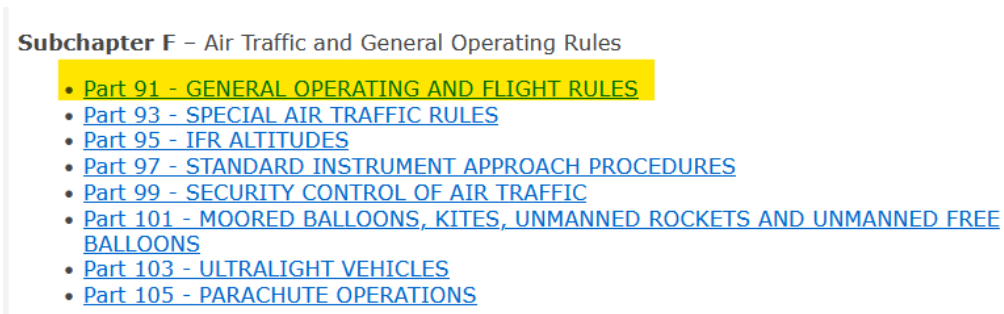
```
NUSC                              0.011396
137                               0.011362
135                               0.008392
121                               0.007639
Part 137: Agricultural            0.004916
UNK                               0.004174
Part 135: Air Taxi & Commuter     0.003352
PUBU                              0.002846
129                               0.002767
Part 121: Air Carrier             0.001856
133                               0.001204
Part 129: Foreign                 0.001125
Non-U.S., Non-Commercial          0.001091
Non-U.S., Commercial              0.001046
Part 133: Rotorcraft Ext. Load    0.000360
Unknown                           0.000247
Public Use                        0.000214
091K                              0.000157
ARMF                              0.000090
Part 125: 20+ Pax,6000+ lbs       0.000056
125                               0.000056
107                               0.000045
103                               0.000022
Public Aircraft                   0.000022
Part 91 Subpart K: Fractional     0.000011
Part 91F: Special Flt Ops.        0.000011
437                               0.000011
Armed Forces                      0.000011
Name: proportion, dtype: float64
```





I interpret that the 091 and Part 91: General Aviation are the same norm of aviation. Basing myself in these research:

https://www.risingup.com/fars/info/



Subchapter F – Air Traffic and General Operating Rules

- Part 91 - GENERAL OPERATING AND FLIGHT RULES
- Part 93 - SPECIAL AIR TRAFFIC RULES
- Part 95 - IFR ALTITUDES
- Part 97 - STANDARD INSTRUMENT APPROACH PROCEDURES
- Part 99 - SECURITY CONTROL OF AIR TRAFFIC
- Part 101 - MOORED BALLOONS, KITES, UNMANNED ROCKETS AND UNMANNED FREE BALLOONS
- Part 103 - ULTRALIGHT VEHICLES
- Part 105 - PARACHUTE OPERATIONS

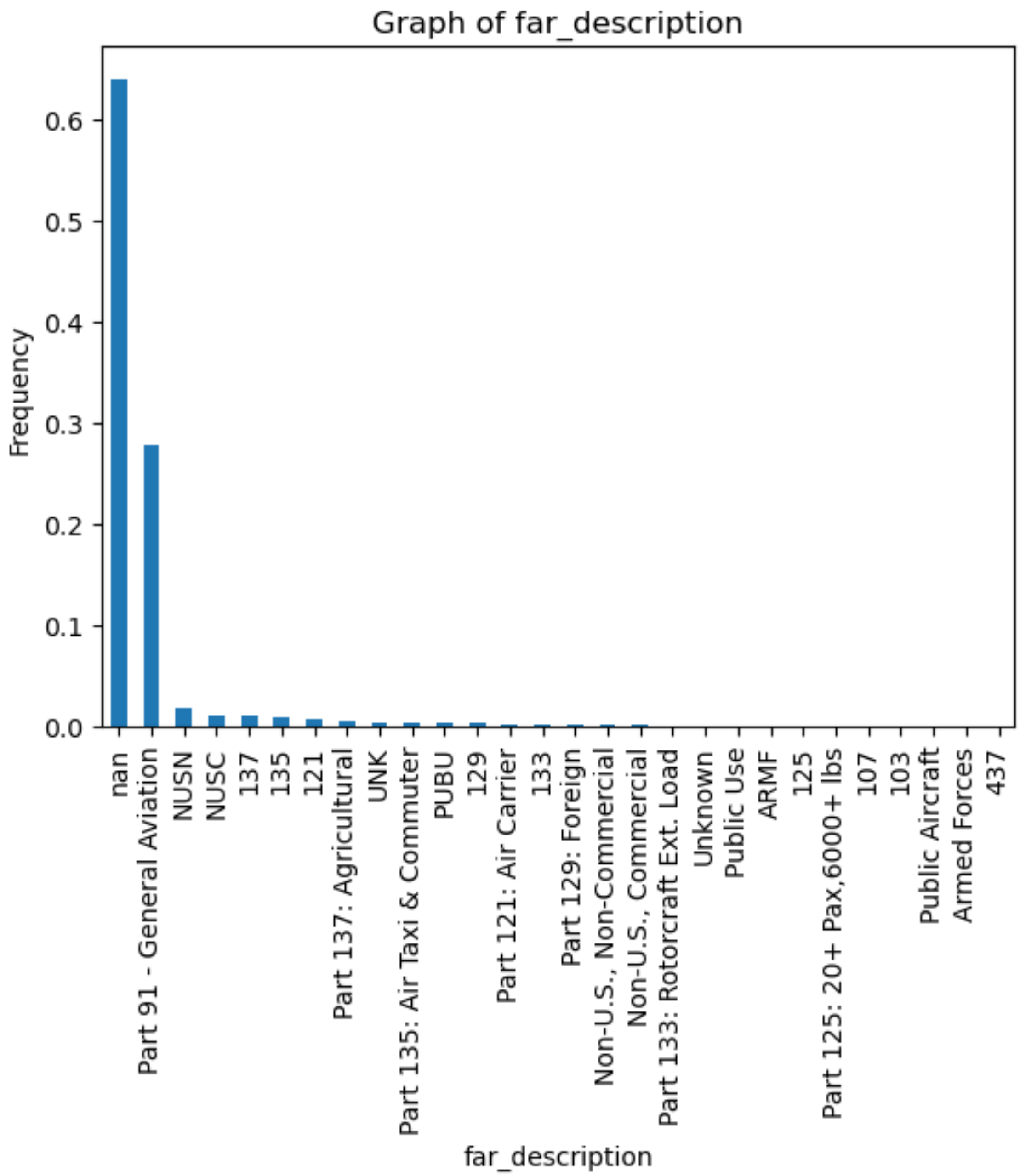
I will proceed to join both of these values and print the result

In [27]:

```
df['far_description'] = df['far_description'].map(lambda x: 'Part 91 - Genera
plot_column_data(df,'far_description', 'bar')
```

```
far_description
NaN                             0.639742
Part 91 - General Aviation      0.278133
NUSN                            0.017820
NUSC                            0.011396
137                             0.011362
135                             0.008392
121                             0.007639
Part 137: Agricultural          0.004916
UNK                             0.004174
Part 135: Air Taxi & Commuter   0.003352
PUBU                            0.002846
129                             0.002767
Part 121: Air Carrier           0.001856
133                             0.001204
Part 129: Foreign               0.001125
Non-U.S., Non-Commercial        0.001091
Non-U.S., Commercial            0.001046
Part 133: Rotorcraft Ext. Load  0.000360
Unknown                         0.000247
Public Use                      0.000214
ARMF                            0.000090
125                             0.000056
Part 125: 20+ Pax,6000+ lbs     0.000056
```

```
107                                  0.000045
103                                  0.000022
Public Aircraft                      0.000022
Armed Forces                         0.000011
437                                  0.000011
Name: proportion, dtype: float64
```



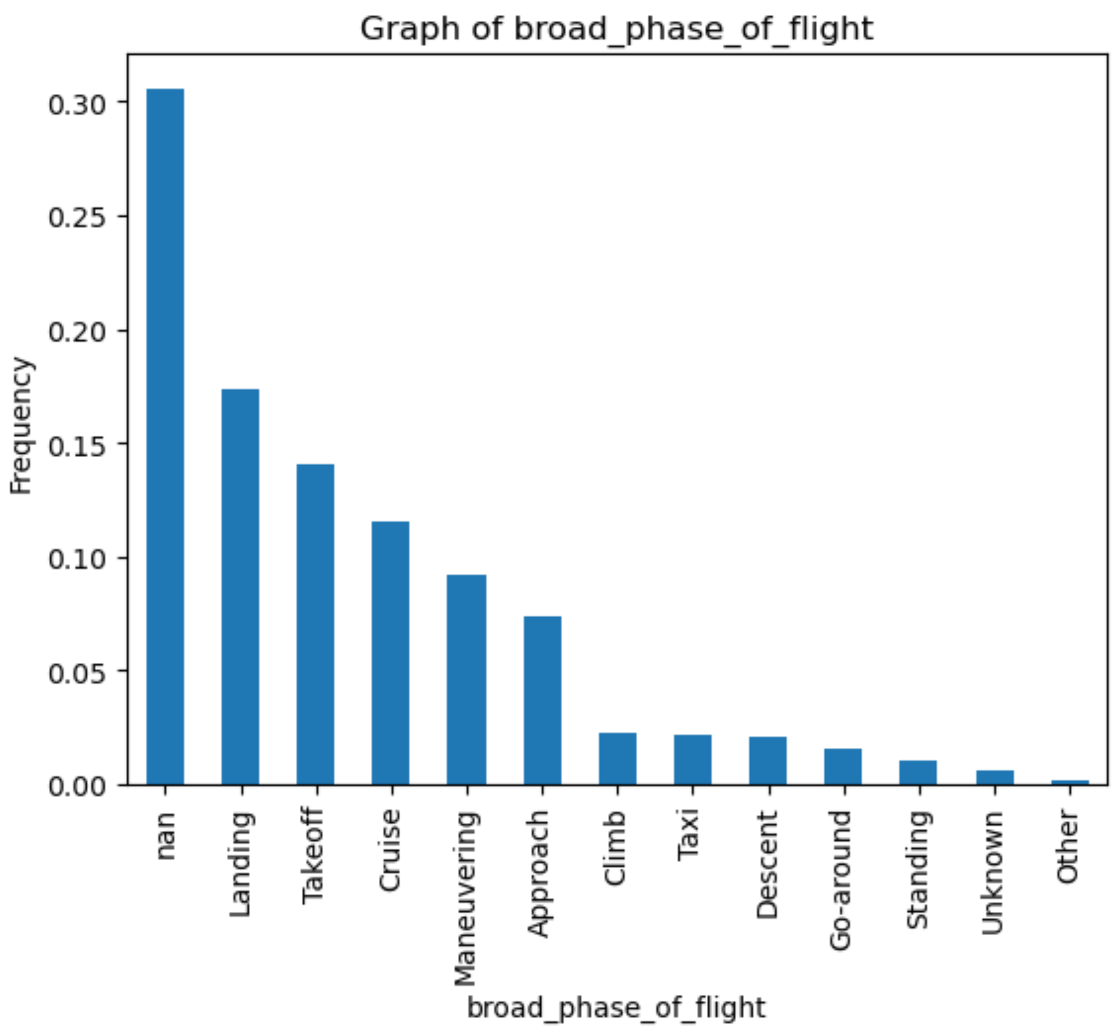
As is visible, the aircrafts under 91 regulations encompass the most part of the dataset about aviation accidents. Given that the far description has 64% of null values, I will add this column to the drop list

In [28]:
```python
drop_columns.append('far_description')
```

Broad Phase of Flight

In [29]:
```
plot_column_data(df,'broad_phase_of_flight', 'bar')
```

```
broad_phase_of_flight
NaN            0.305606
Landing        0.173565
Takeoff        0.140546
Cruise         0.115526
Maneuvering    0.091620
Approach       0.073642
Climb          0.022882
Taxi           0.022027
Descent        0.021229
Go-around      0.015221
Standing       0.010631
Unknown        0.006165
Other          0.001339
Name: proportion, dtype: float64
```



The most important causes of accidents happened either during: landing, takeoff, cruise, maneuvering or approach. I consider 30% of null values to not be too excesive and believe that the 5 phases mentioned before could be of use. I will not drop these columns

In [30]:
```
df['broad_phase_of_flight'].fillna('Unknown', inplace=True)
```

The study of the columns in question have been done and I will now proceed to drop said columns. I will also append to the drop columns the previous id columns that are now unnecessary with the new primary_key column

In [31]:
```
drop_columns = drop_columns + ['accident_number', 'registration_number', 'eve

df = df.drop(drop_columns, axis=1)
```

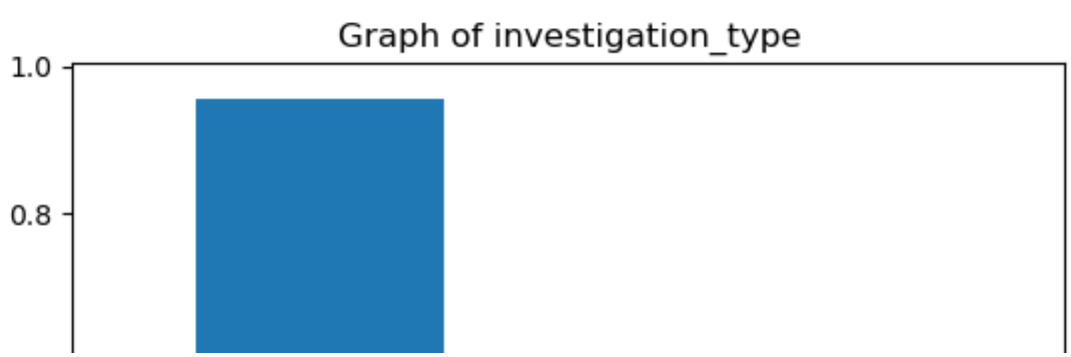
In [32]:
```
df.isnull().sum()/len(df)*100
```

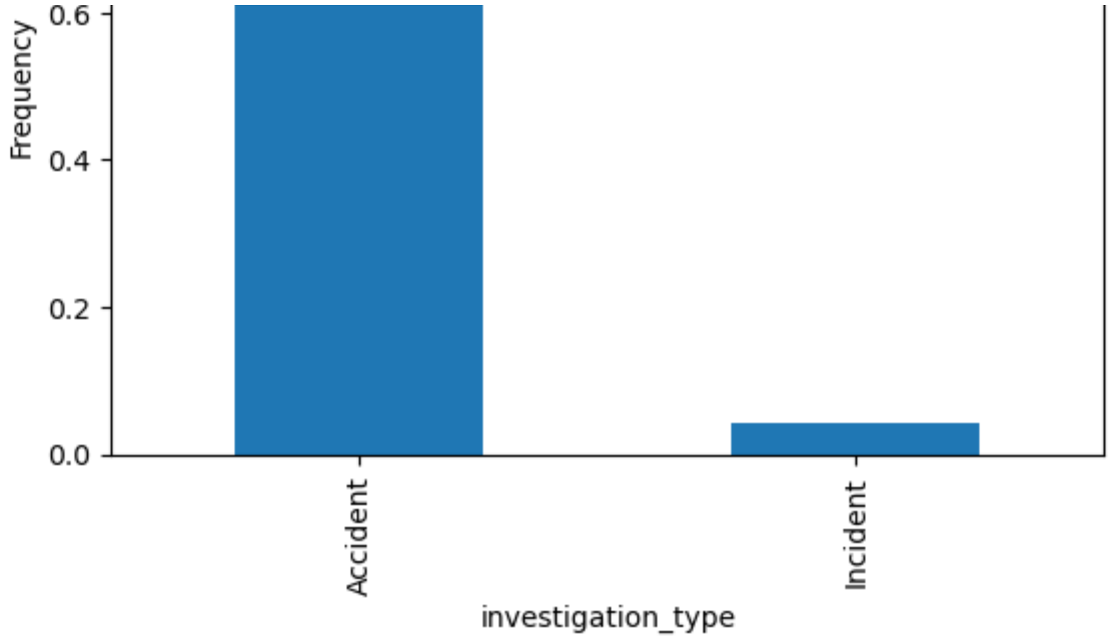
Out[32]:
```
investigation_type         0.000000
event_date                 0.000000
location                   0.058500
country                    0.254250
injury_severity            1.124999
aircraft_damage            3.593246
make                       0.070875
model                      0.103500
amateur_built              0.114750
number_of_engines          6.844491
engine_type                7.982990
purpose_of_flight          6.965991
total_fatal_injuries      12.826109
total_serious_injuries    14.073732
total_minor_injuries      13.424608
total_uninjured            6.650992
weather_condition          5.053494
broad_phase_of_flight      0.000000
report_status              7.181991
primary_key                1.554748
dtype: float64
```

Investigation type

In [33]:
```
plot_column_data(df,'investigation_type', 'bar')
```

```
investigation_type
Accident    0.956418
Incident    0.043582
Name: proportion, dtype: float64
```





After doing some reasearch, we have noticed that an accident is a unintentional event that results in harm whereas an incident although it might be unintentional doesn't necessarily result in harm

Moreover, we can see that all of the registrations in the dataset are all accidents (in 96% of it's totality)

Event Date

In [34]:
```
df['event_date'].min()
```

Out[34]:
```
'1948-10-24'
```

In [35]:
```
df['event_date'].max()
```

Out[35]:
```
'2022-12-29'
```

In [36]:
```
df['event_date'].value_counts(normalize=True, dropna=False)
```

Out[36]:
```
event_date
1984-06-30    0.000281
1982-05-16    0.000281
2000-07-08    0.000281
1983-08-05    0.000270
1984-08-25    0.000270
                 ...
2014-03-16    0.000011
2014-03-15    0.000011
2014-03-12    0.000011
2014-03-10    0.000011
2022-12-29    0.000011
Name: proportion, Length: 14782, dtype: float64
```

In [37]:
```
df['event_date'].isna().any()
```

Out[37]:
```
False
```

We have realized that we have accidents or incidents from 1948 to 2022

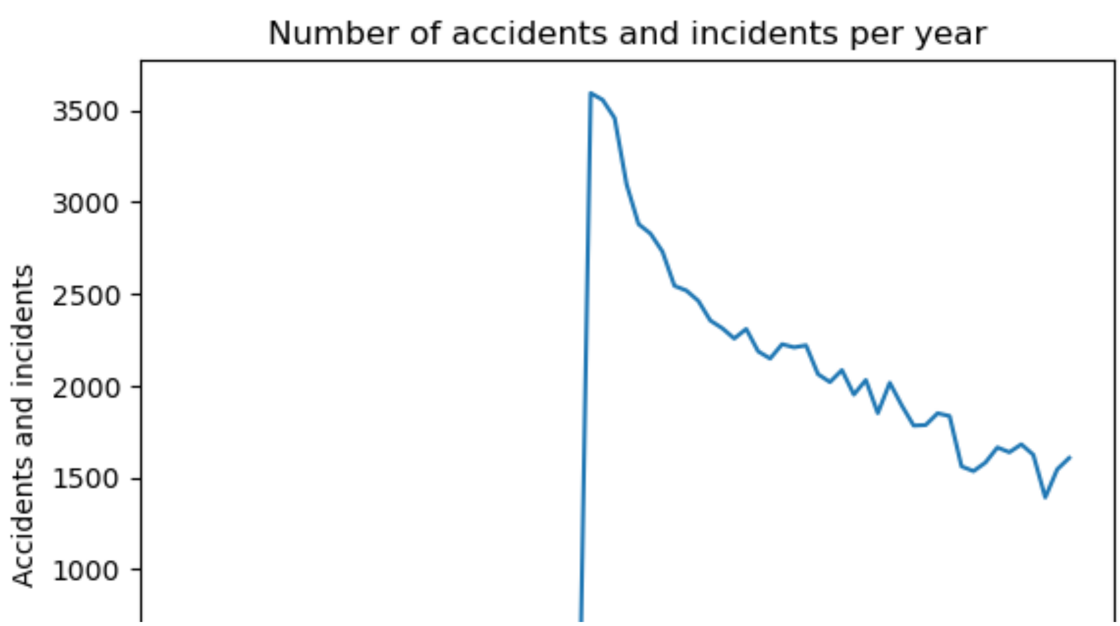
In [38]:
```
df['event_date'] = df['event_date'].astype('object')
```

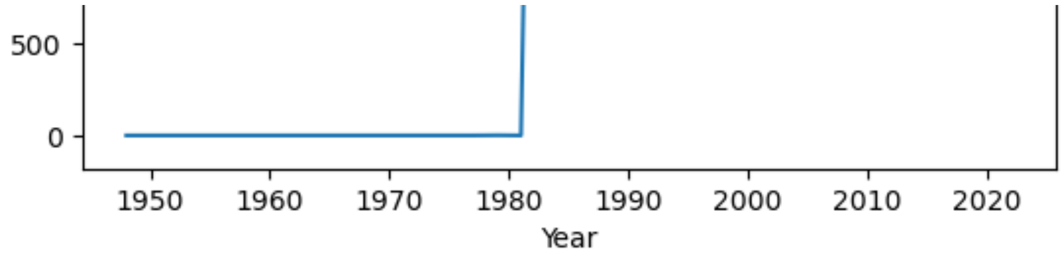
I would like to investigate the number of accidents per year and month

In [39]:
```
df['year'] = df['event_date'].map(lambda x:int(x[:4]))
df['year']
```

Out[39]:
```
0        1948
1        1962
2        1974
3        1977
4        1979
         ...
88884    2022
88885    2022
88886    2022
88887    2022
88888    2022
Name: year, Length: 88889, dtype: int64
```

In [40]:
```
df.groupby('year')['investigation_type'].count().plot(kind='line')

plt.title('Number of accidents and incidents per year')
plt.xlabel('Year')
plt.ylabel('Accidents and incidents')

plt.show()
```





In [41]:

```
df[df['year']<1982]
```

Out[41]:

	investigation_type	event_date	location	country	injury_severity	aircraft_dam
0	Accident	1948-10-24	MOOSE CREEK, ID	United States	Fatal(2)	Destr
1	Accident	1962-07-19	BRIDGEPORT, CA	United States	Fatal(4)	Destr
2	Accident	1974-08-30	Saltville, VA	United States	Fatal(3)	Destr
3	Accident	1977-06-19	EUREKA, CA	United States	Fatal(2)	Destr
4	Accident	1979-08-02	Canton, OH	United States	Fatal(1)	Destr
5	Accident	1979-09-17	BOSTON, MA	United States	Non-Fatal	Substa
6	Accident	1981-08-01	COTTON, MN	United States	Fatal(4)	Destr

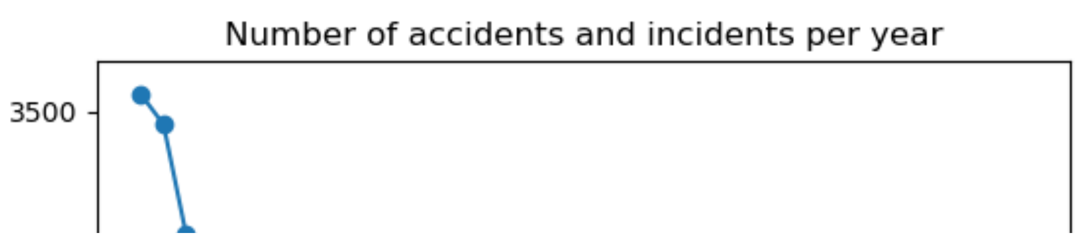
As can be seen there is only 7 rows of data before 1983. I will proceed to eliminate these rows

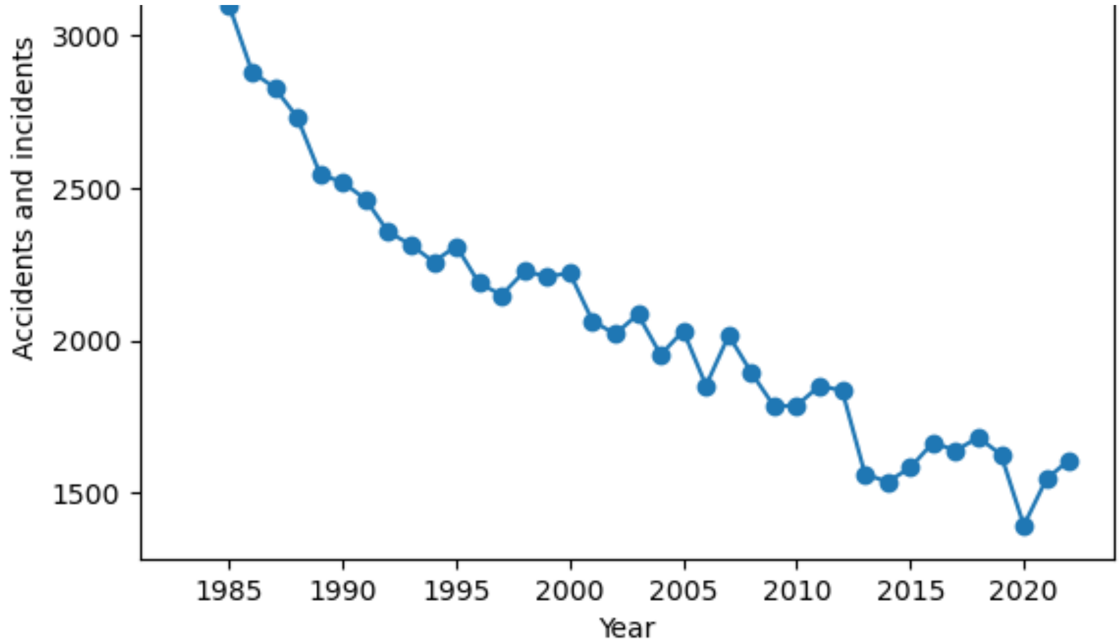
In [42]:

```
df = df[df['year']>1982]
```

In [43]:

```
df.groupby('year')['investigation_type'].count().plot(kind='line', marker='o

plt.title('Number of accidents and incidents per year')
plt.xlabel('Year')
plt.ylabel('Accidents and incidents')

plt.show()
```





In the passing of time, it is visible that the number of accidents have reduced gradually. In 2020, there is a noticeable drop in the number of accidents, possible due to the Covid-19 restrictions period

I will now study the number of accidents per month

In [44]:

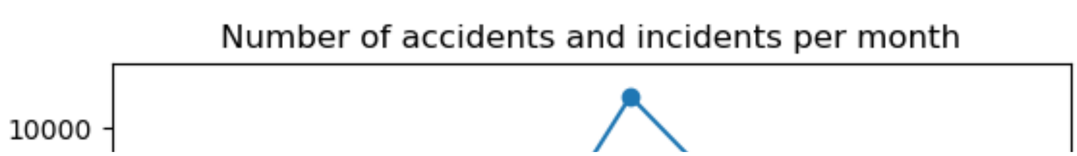
```
df['month'] = df['event_date'].map(lambda x:x[5:7])
df['month']
```

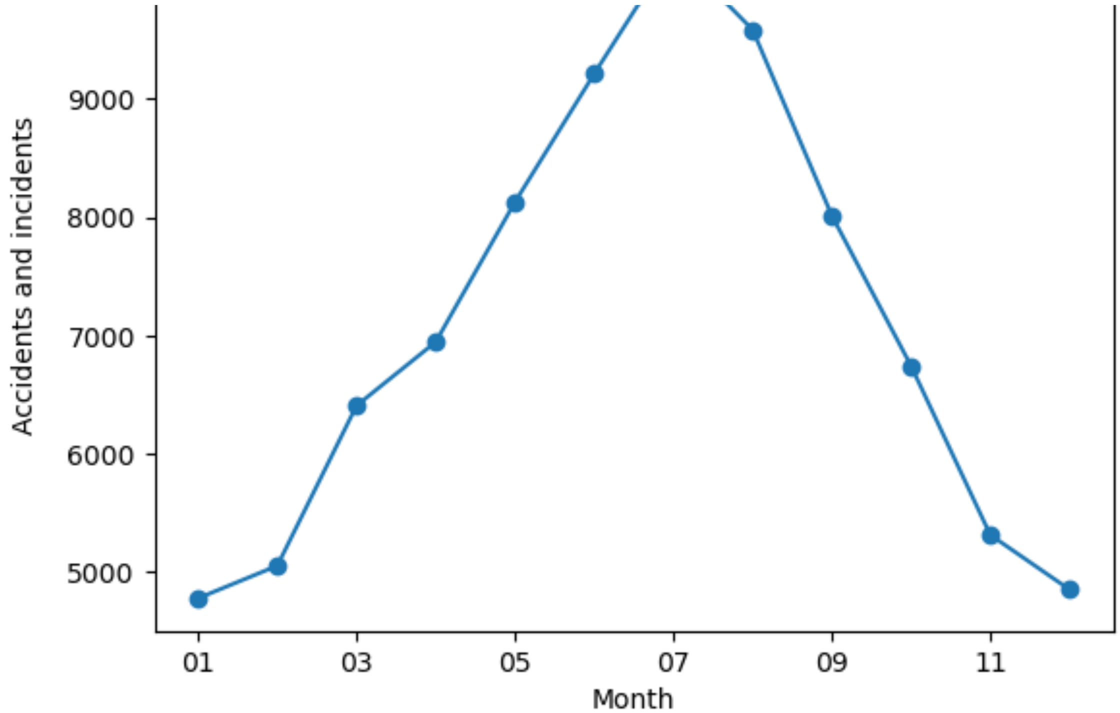
Out[44]:

```
3600     01
3601     01
3602     01
3603     01
3604     01
         ..
88884    12
88885    12
88886    12
88887    12
88888    12
Name: month, Length: 85289, dtype: object
```

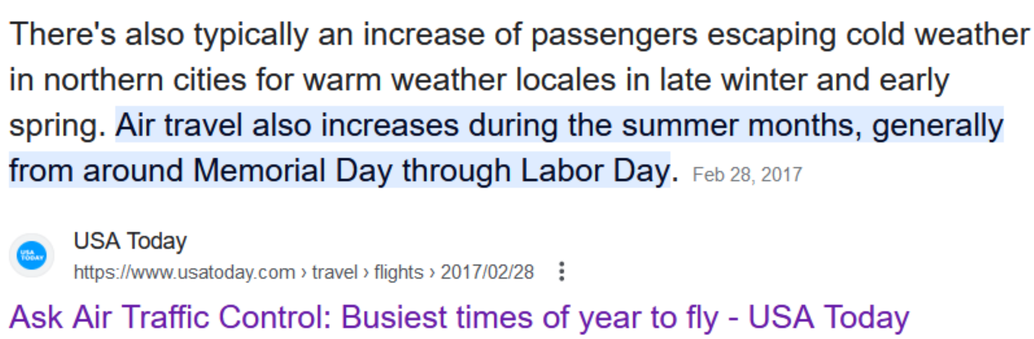
In [45]:

```
df.groupby('month')['investigation_type'].count().plot(kind='line', marker='(

plt.title('Number of accidents and incidents per month')
plt.xlabel('Month')
plt.ylabel('Accidents and incidents')

plt.show()
```





The information shows that there are more accidents and incidents during the summer period which is normal as there tends to be more flights during that period as can be seen in the following studies:


There's also typically an increase of passengers escaping cold weather in northern cities for warm weather locales in late winter and early spring. Air travel also increases during the summer months, generally from around Memorial Day through Labor Day. Feb 28, 2017

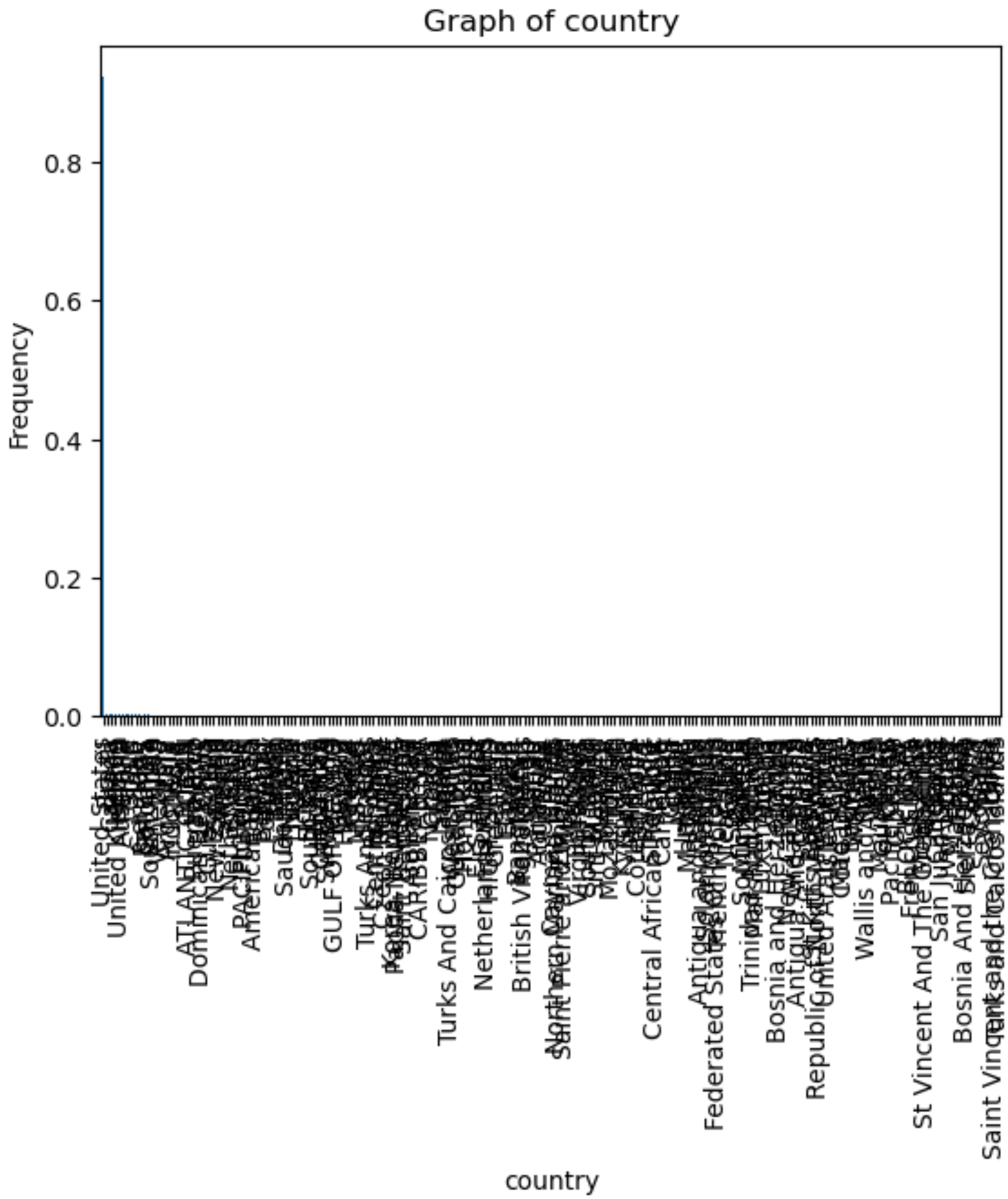
USA Today
https://www.usatoday.com › travel › flights › 2017/02/28

Ask Air Traffic Control: Busiest times of year to fly - USA Today

Country

In [46]:
```
plot_column_data(df,'country', 'bar')
```

```
country
United States                       0.922475
Brazil                              0.004385
Canada                              0.004209
Mexico                              0.004197
United Kingdom                      0.004033
                                      ...
Seychelles                          0.000012
Palau                               0.000012
Libya                               0.000012
Saint Vincent and the Grenadines    0.000012
Turks and Caicos Islands            0.000012
Name: proportion, Length: 220, dtype: float64
```



The majority of the events occur in USA. I will delete the rows where the country is not USA

In [47]:
```
df = df[df['country']=='United States']
```

Injury severity

In [48]:
```
df['injury_severity'].value_counts(normalize=True, dropna=False)
```

Out[48]:
```
injury_severity
Non-Fatal      0.788896
Fatal(1)       0.070656
Fatal          0.045197
```

```
Fatal          0.049197
Fatal(2)       0.041334
Incident       0.022294
Fatal(3)       0.012024
Fatal(4)       0.008262
Minor          0.002580
Fatal(5)       0.002110
Serious        0.001945
NaN            0.001373
Fatal(6)       0.001335
Fatal(7)       0.000432
Fatal(8)       0.000280
Fatal(10)      0.000216
Unavailable    0.000191
Fatal(9)       0.000102
Fatal(14)      0.000064
Fatal(11)      0.000064
Fatal(12)      0.000051
Fatal(17)      0.000038
Fatal(13)      0.000038
Fatal(18)      0.000038
Fatal(25)      0.000038
Fatal(82)      0.000025
Fatal(23)      0.000025
Fatal(20)      0.000025
Fatal(34)      0.000025
Fatal(31)      0.000013
Fatal(65)      0.000013
Fatal(19)      0.000013
Fatal(44)      0.000013
Fatal(64)      0.000013
Fatal(21)      0.000013
Fatal(92)      0.000013
Fatal(265)     0.000013
Fatal(228)     0.000013
Fatal(49)      0.000013
Fatal(70)      0.000013
Fatal(88)      0.000013
Fatal(15)      0.000013
Fatal(29)      0.000013
Fatal(230)     0.000013
Fatal(110)     0.000013
Fatal(68)      0.000013
Fatal(132)     0.000013
Fatal(37)      0.000013
Fatal(16)      0.000013
Fatal(135)     0.000013
Fatal(73)      0.000013
Fatal(111)     0.000013
Fatal(43)      0.000013
Fatal(28)      0.000013
Fatal(156)     0.000013
Fatal(27)      0.000013
Name: proportion, dtype: float64
```

In [49]:

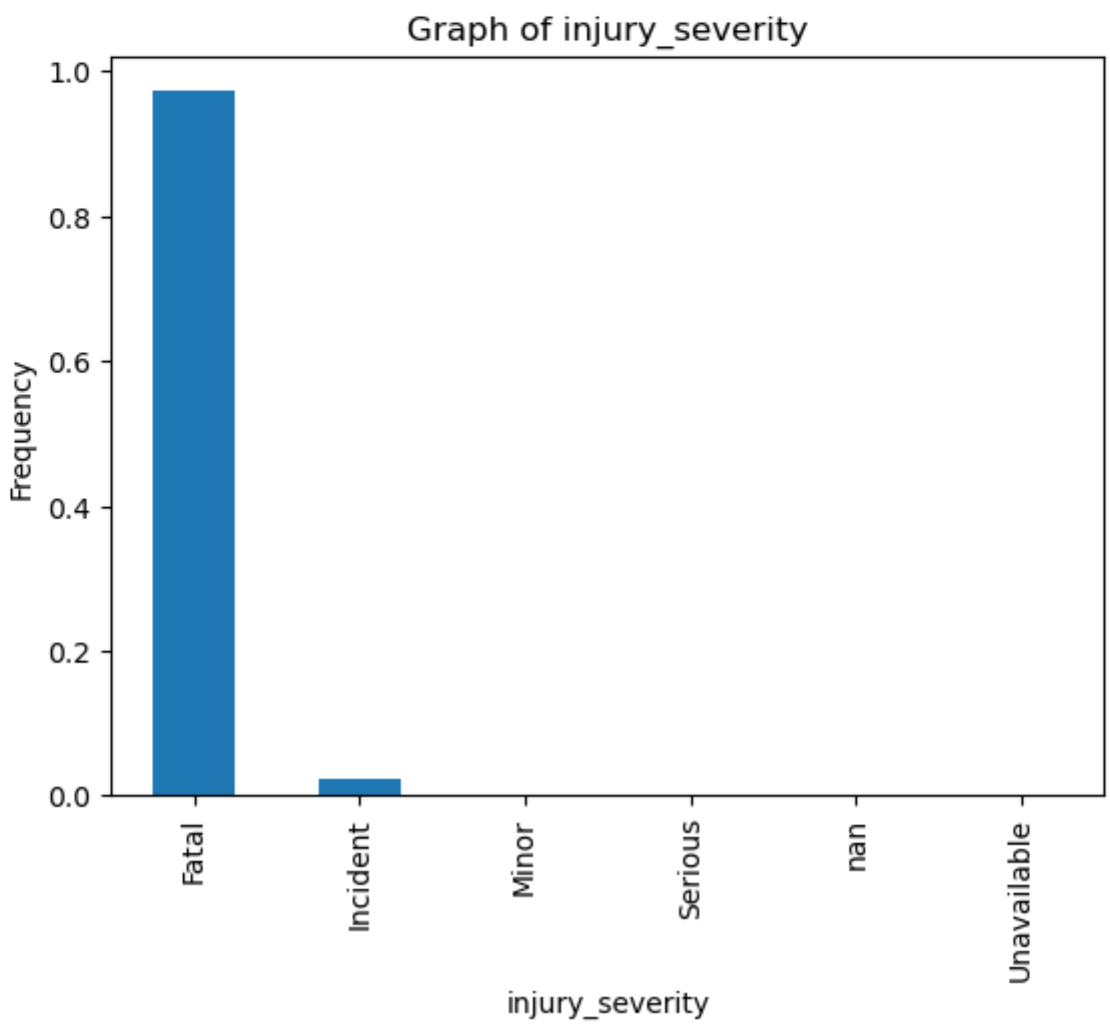
```
df['injury_severity'] = df['injury_severity'].astype('category')
```

I am going to group all the Fatal injuries. First I'll change the type of the column to categorical

In [50]:
```
df['injury_severity'] = df['injury_severity'].map(lambda x: 'Fatal' if isinst
```

In [51]:
```
plot_column_data(df,'injury_severity', 'bar')
```

```
injury_severity
Fatal          0.971618
Incident       0.022294
Minor          0.002580
Serious        0.001945
NaN            0.001373
Unavailable    0.000191
Name: proportion, dtype: float64
```

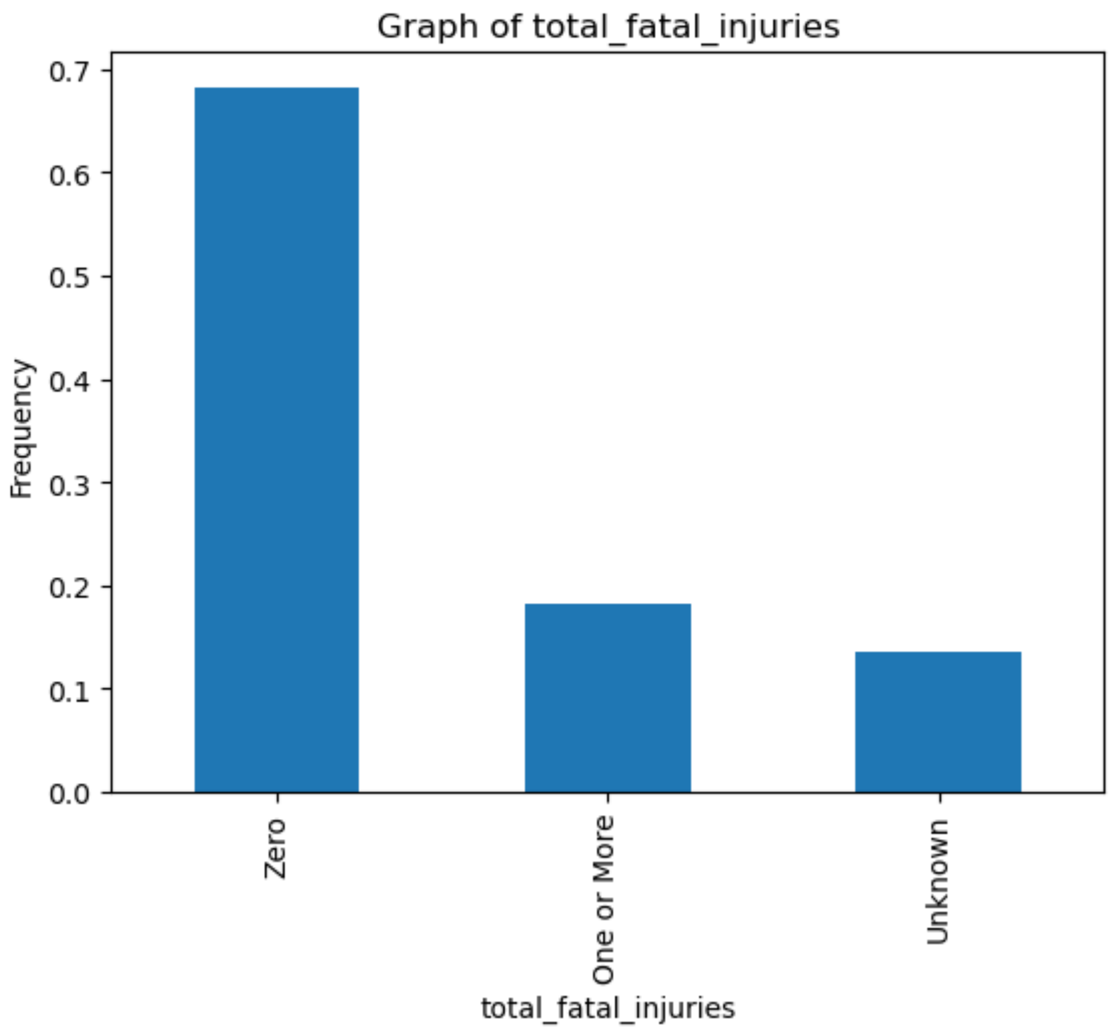


'total_fatal_injuries', 'total_serious_injuries', 'total_minor_injuries', 'total_uninjured' looking at their frequencies

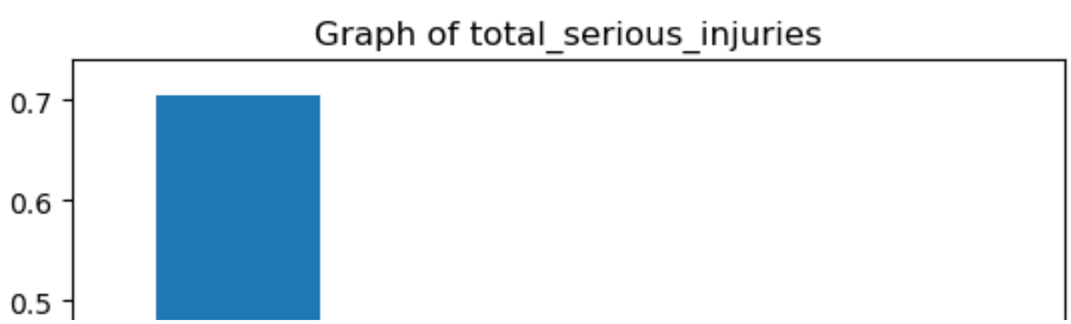
In [52]:
```
columns_of_injuries = ['total_fatal_injuries', 'total_serious_injuries', 'tot

for columns in columns_of_injuries:
    # First, I am going to call categorize_data function to categorize the va
```

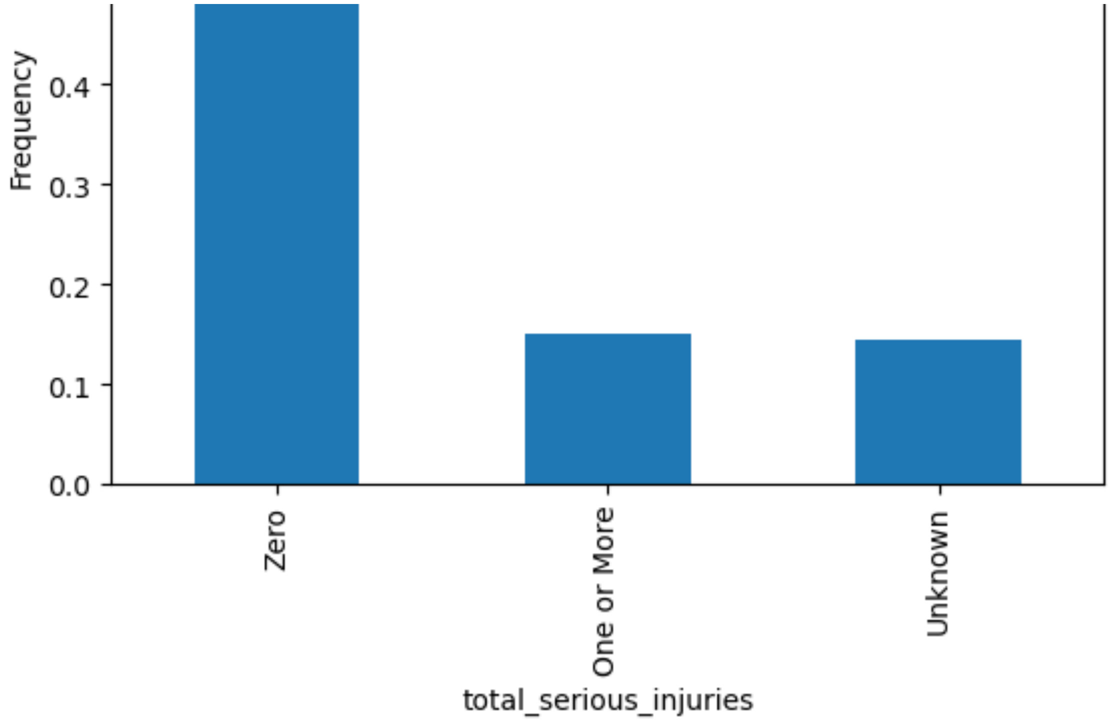
```
    df[columns] = df[columns].map(categorize_data)

    # Second, I will represent the results of all the columns in bar graphs c
    plot_column_data(df,columns, 'bar')
```

```
total_fatal_injuries
Zero           0.682220
One or More    0.182684
Unknown        0.135097
Name: proportion, dtype: float64
```

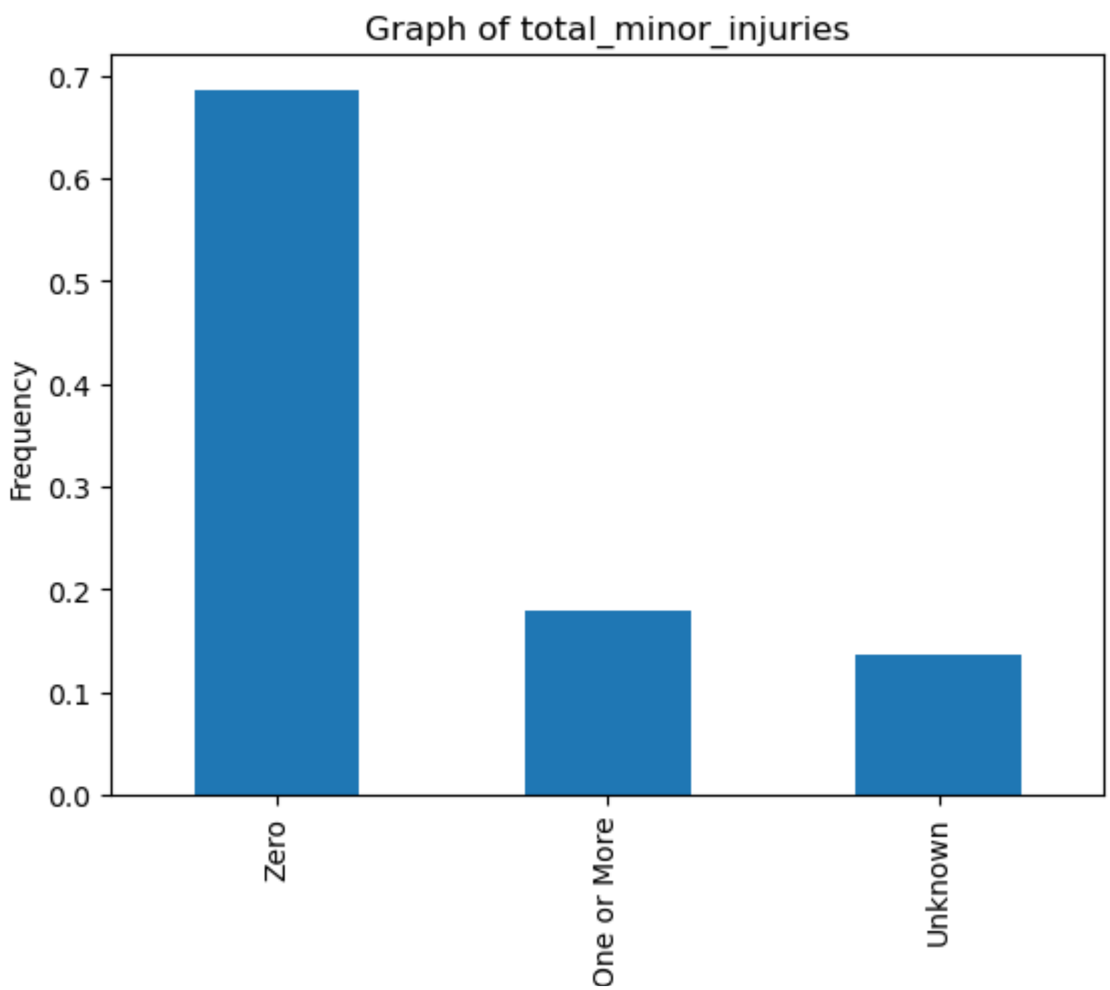



```
total_serious_injuries
Zero           0.705009
One or More    0.150781
Unknown        0.144210
Name: proportion, dtype: float64
```



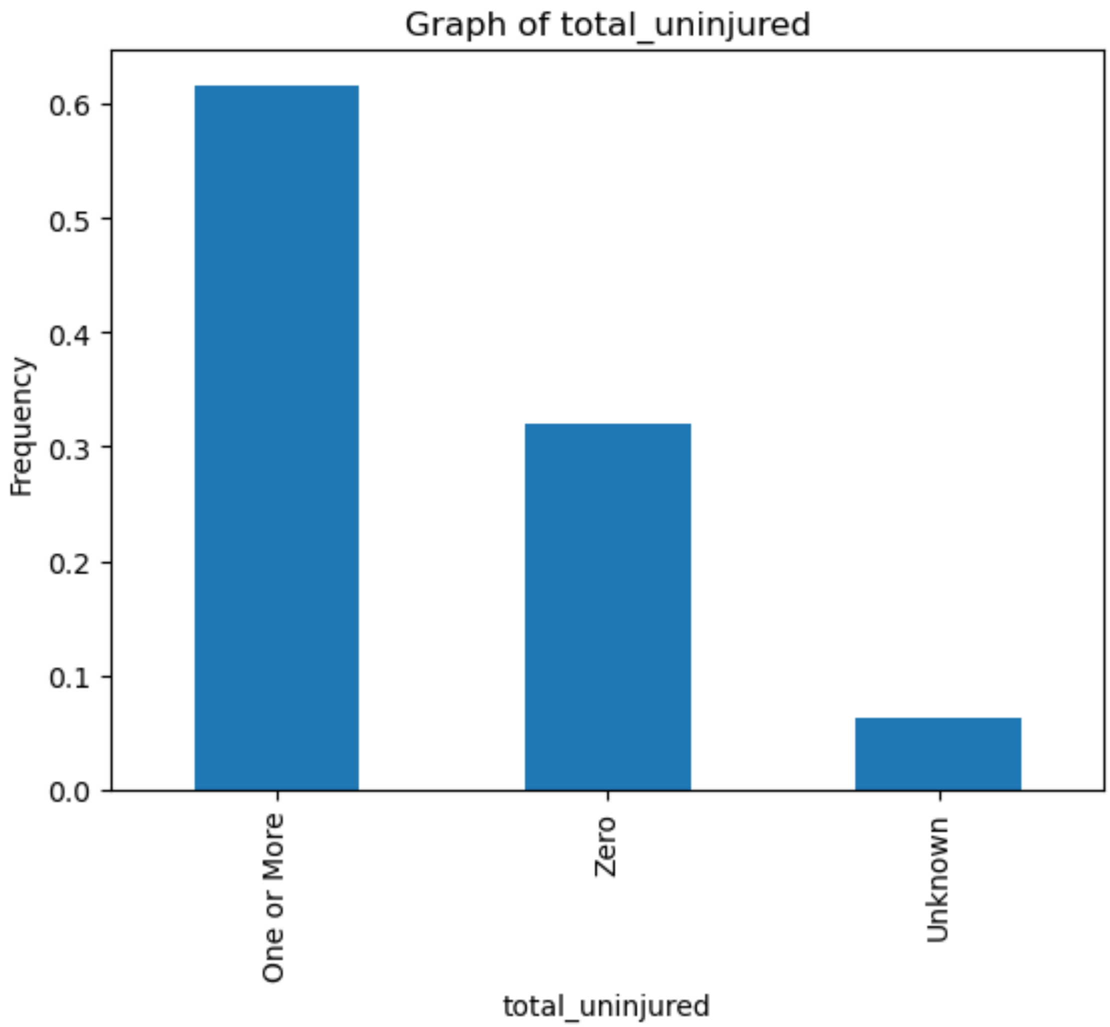



```
total_minor_injuries
Zero           0.685639
One or More    0.178349
Unknown        0.136012
Name: proportion, dtype: float64
```



total_minor_injuries

```
total_uninjured
One or More    0.615707
Zero           0.320793
Unknown        0.063500
Name: proportion, dtype: float64
```



The graphs above give a view of the injuries. In particular, in the value counts one can see that around 30% of the accidents in the dataset have had injuries

I will eliminate the rows where injury_severity has 'Fatal' but don't have a number in the corresponding value of total_fatal_injuries

In [53]:
```
df = df[~((df['injury_severity']=='Fatal') & (df['total_fatal_injuries']=='Ze
```

I will eliminate the rows where injury_severity has 'Non-Fatal' but have a number in the corresponding value of total_fatal_injuries

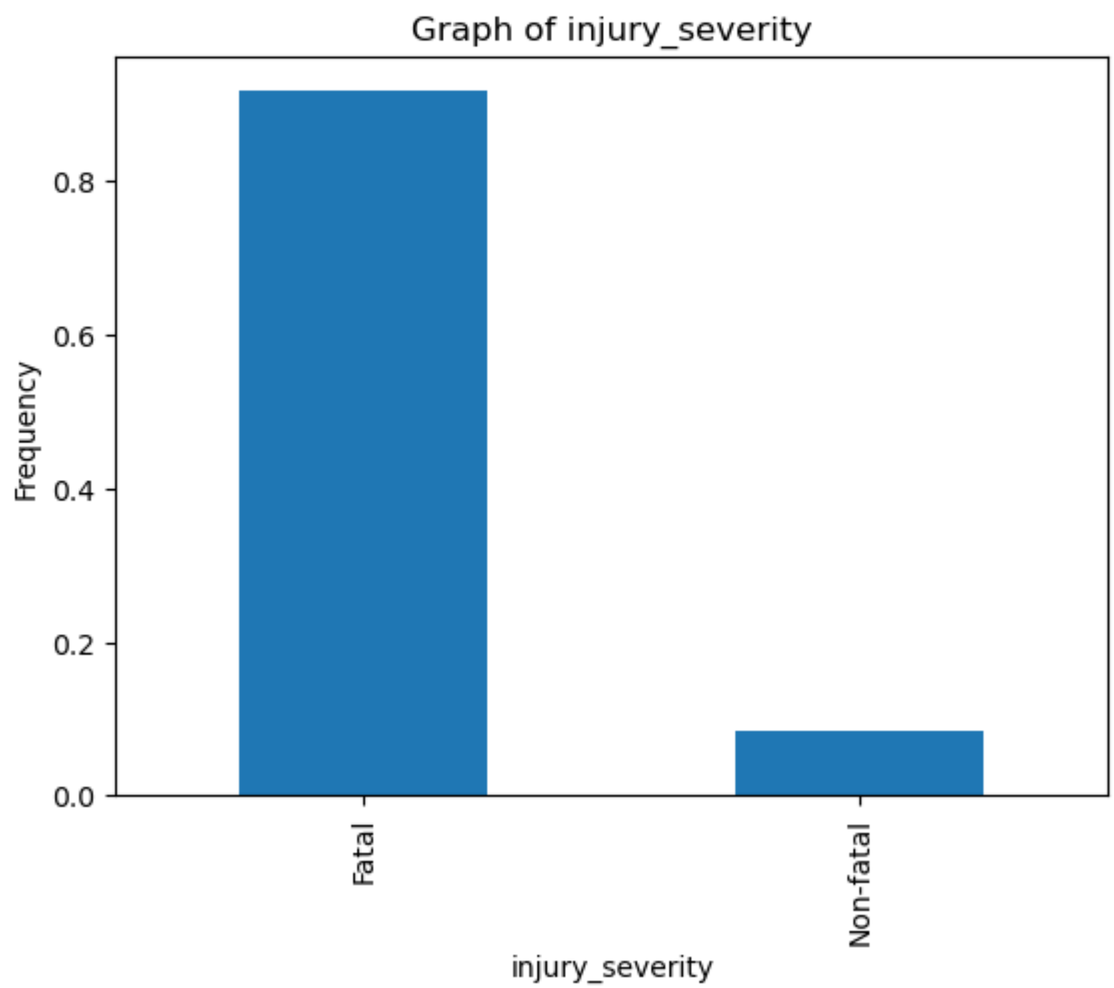
In [54]:
```
df = df[~((df['injury_severity']=='Non-Fatal') & (df['total_fatal_injuries']
```

Given that it is of my interest to study the fatal injuries above all, I will categorize injury_severity to either fatal or non-fatal.

In [55]:

```
df['injury_severity'] = df['injury_severity'].map(lambda x: 'Non-fatal' if x
plot_column_data(df,'injury_severity', 'bar')
```

```
injury_severity
Fatal        0.916747
Non-fatal    0.083253
Name: proportion, dtype: float64

<>:1: SyntaxWarning: "is not" with 'str' literal. Did you mean "!="?
<>:1: SyntaxWarning: "is not" with 'str' literal. Did you mean "!="?
C:\Users\Usuario\AppData\Local\Temp\ipykernel_17664\588282316.py:1: SyntaxWarn
ing: "is not" with 'str' literal. Did you mean "!="?
  df['injury_severity'] = df['injury_severity'].map(lambda x: 'Non-fatal' if x
is not 'Fatal' else x)
```



I decide to select the registrations of total fatal injuries and of total serious injuries that are 'One or More' and study those from now onwards. We don't consider minor injuries because they might be negligible

In [56]:

```
df = df[(df['total_fatal_injuries']=='One or More') | (df['total_serious_inju
```

In [57]:
```python
df = df.reset_index(drop=True)
df
```

Out[57]:

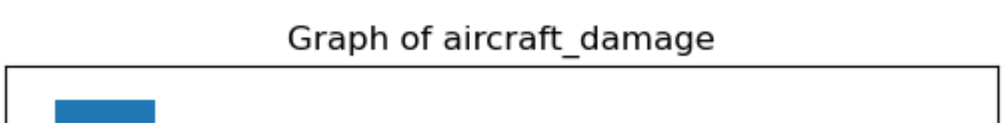
	investigation_type	event_date	location	country	injury_severity	aircraft_
0	Accident	1983-01-02	GENOA CITY, WI	United States	Fatal	D
1	Accident	1983-01-02	BEAUFORT, SC	United States	Fatal	D
2	Accident	1983-01-02	HANCOCK, MD	United States	Fatal	Su
3	Accident	1983-01-03	WILLARD, WA	United States	Fatal	D
4	Accident	1983-01-03	AVALON, CA	United States	Fatal	D
...	...	...	...	...	...	
16048	Accident	2022-12-17	Cottonwood, CA	United States	Non-fatal	
16049	Accident	2022-12-21	Auburn Hills, MI	United States	Non-fatal	
16050	Accident	2022-12-21	Reserve, LA	United States	Non-fatal	
16051	Accident	2022-12-26	Annapolis, MD	United States	Non-fatal	
16052	Accident	2022-12-29	Athens, GA	United States	Non-fatal	

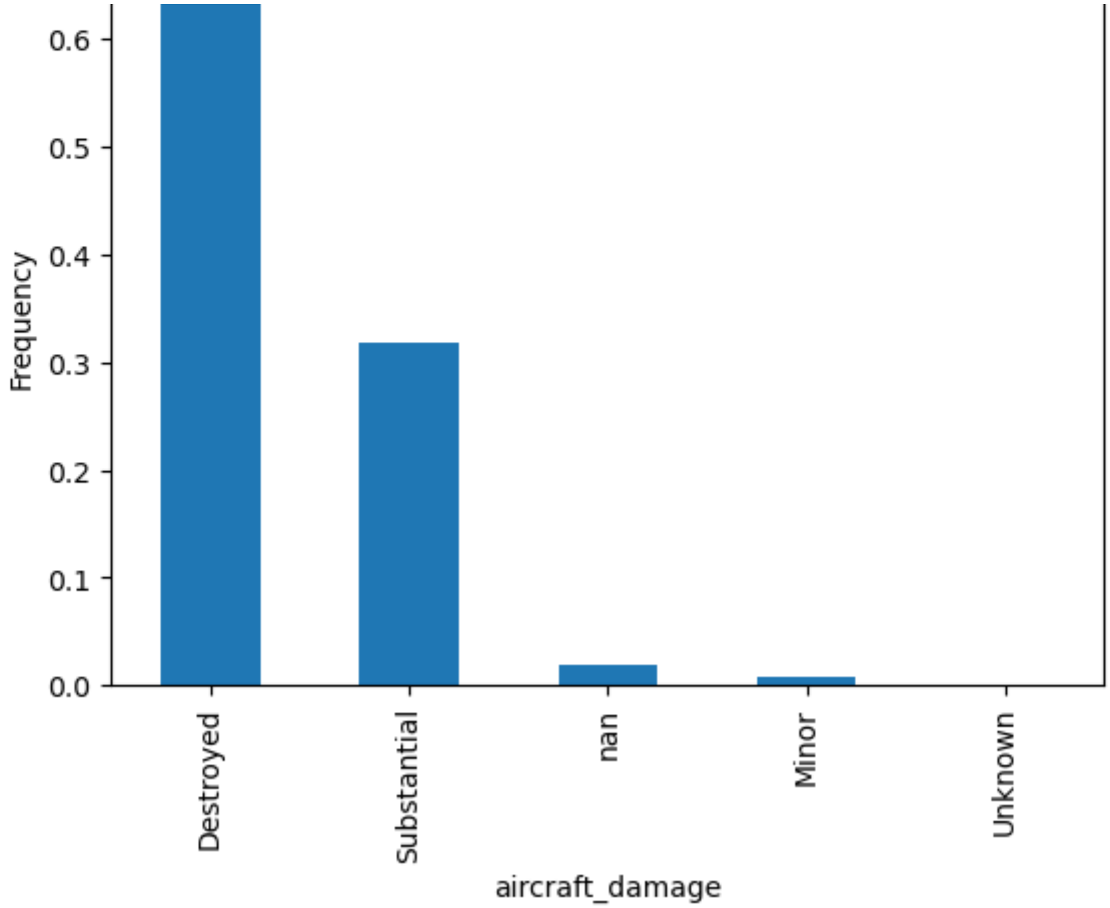
16053 rows × 22 columns

Aircraft Damage

In [58]:
```python
plot_column_data(df,'aircraft_damage', 'bar')
```

```
aircraft_damage
Destroyed      0.652900
Substantial    0.317635
NaN            0.019934
Minor          0.008846
Unknown        0.000685
Name: proportion, dtype: float64
```





In [59]:
```
df['aircraft_damage'].fillna('Unknown', inplace=True)
```

As we can see the majority of the damages are substantial and destroyed. I will proceed with further investigations

Make

In [60]:
```
df['make'].value_counts(normalize=True, dropna=False)
```

Out[60]:
```
make
Cessna                  0.211985
Piper                   0.148695
Beech                   0.073133
CESSNA                  0.040304
PIPER                   0.027970
                          ...
Bensen Aircraft Corp.   0.000062
Boykin B J              0.000062
Motley Vans             0.000062
Madsen                  0.000062
ROYSE RALPH L           0.000062
Name: proportion, Length: 2846, dtype: float64
```

In [61]:
```
df['make'].isna().sum()
```

Out[61]: 3

In [62]:

```
# Replace null values with 'Unknown'
df['make'].fillna('Unknown', inplace=True)
```

I will change the values of the make column to lower case letters and ensure they're all grouped correctly

In [63]:

```
df['make'] = df['make'].str.capitalize()
```

I am going to joint Douglas with Mcdonnell douglas because they're the same aircraft company

In [64]:

```
df['make'] = df['make'].map(lambda x: 'Douglas' if x in ['Mcdonnell douglas'
```

I will proceed to create a list of the top 10 makers with accidents

In [65]:

```
top_10_make = df['make'].value_counts(normalize=True, dropna=False).head(10)
top_10_make
```

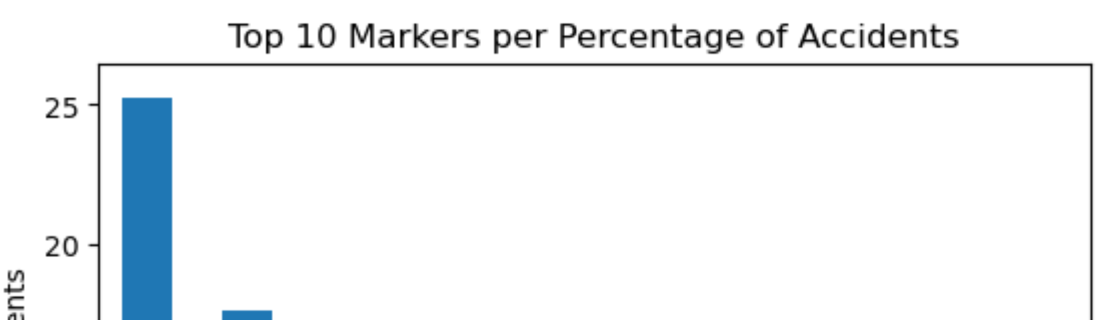
Out[65]:

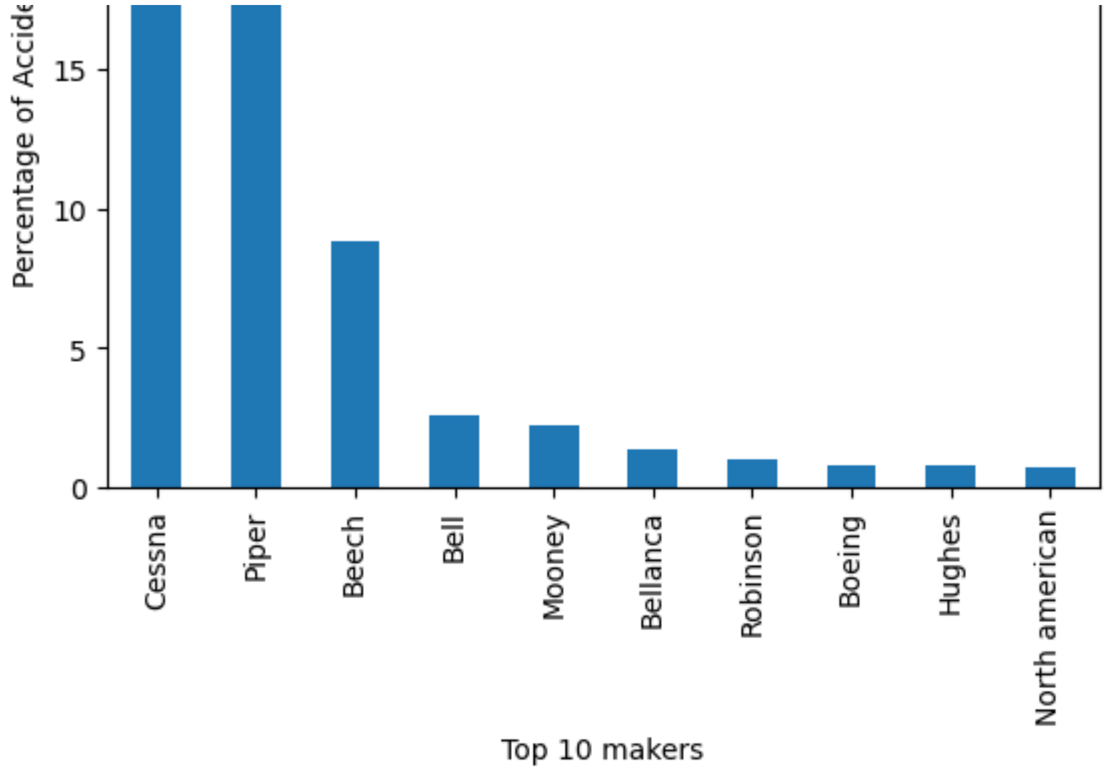
```
make
Cessna              25.228929
Piper               17.666480
Beech                8.851928
Bell                 2.578957
Mooney               2.230113
Bellanca             1.345543
Robinson             1.002928
Boeing               0.822276
Hughes               0.797359
North american       0.735065
Name: proportion, dtype: float64
```

In [66]:

```
plt.figure()
top_10_make.plot(kind='bar')
plt.xlabel('Top 10 makers')
plt.ylabel('Percentage of Accidents')
plt.xticks(rotation=90)
plt.title('Top 10 Markers per Percentage of Accidents')
```

Out[66]: Text(0.5, 1.0, 'Top 10 Markers per Percentage of Accidents')





The above makers are the ones with the highest number of accidents. It's noticeable that Cessna, Piper and Beech are the highest of all

Model

In [67]:

```
df['model'].value_counts(normalize=True, dropna=False)
```

Out[67]:

```
model
152          0.014764
172N         0.011774
PA-28-140    0.011026
A36          0.009095
172          0.008908
               ...
LJ-60        0.000062
172 F        0.000062
DN-1         0.000062
L-39CT       0.000062
EC 130 T2    0.000062
Name: proportion, Length: 4448, dtype: float64
```

In [68]:

```
df['model'].isna().sum()
```

Out[68]:

```
7
```

I don't believe to be able to extract much information from the model column

Amateur Built

In [69]:

```
df['amateur_built'].value_counts(normalize=True, dropna=False)
```

Out[69]:
```
amateur_built
No      0.846633
Yes     0.152869
NaN     0.000498
Name: proportion, dtype: float64
```

As can be seen most of the accident cases were commercial trips

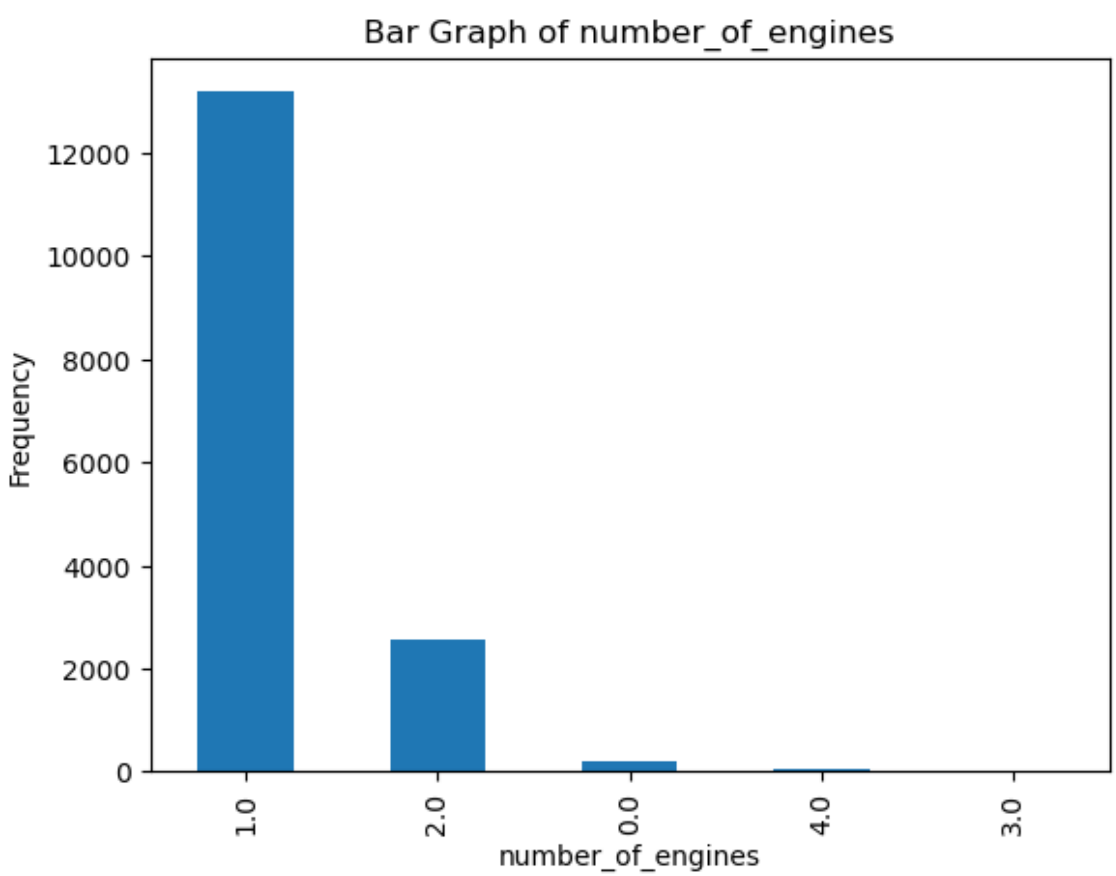
Number of engines

In [70]:
```
df['number_of_engines'].value_counts(normalize=True, dropna=False)
```

Out[70]:
```
number_of_engines
1.0    0.798854
2.0    0.160780
NaN    0.022488
0.0    0.013269
4.0    0.002990
3.0    0.001620
Name: proportion, dtype: float64
```

In [71]:
```
# I will proceed to replace the null values with the mode
df['number_of_engines'].fillna(df['number_of_engines'].mode()[0], inplace=Tru
```

In [72]:
```
plot_bar_graph_for_columns('number_of_engines')
```

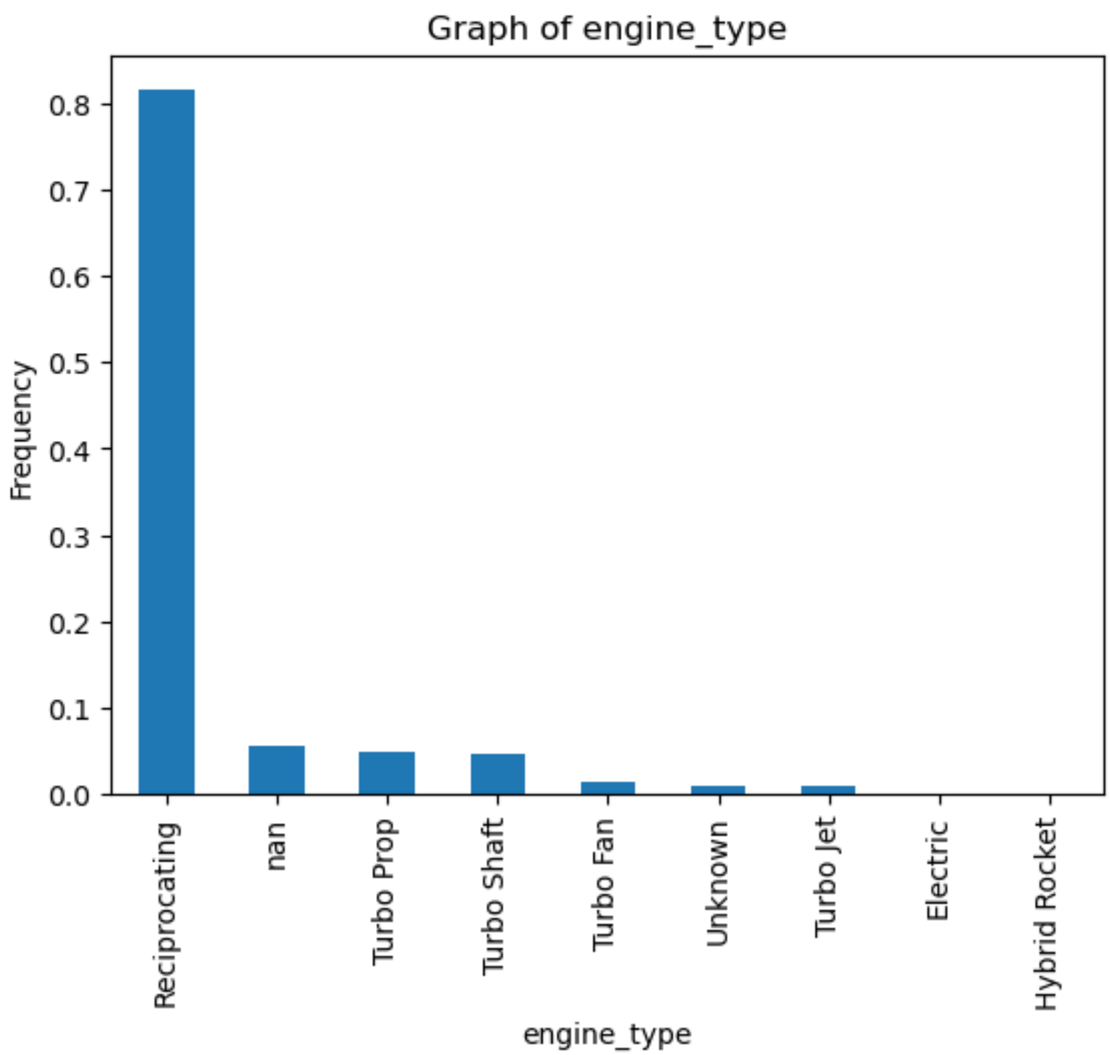


As is clearly visible, the majority of the accidents (in an 80% of the cases) happened with aircrafts that only had one engine

Engine type

In [73]:
```
plot_column_data(df,'engine_type', 'bar')
```

```
engine_type
Reciprocating    0.814365
NaN              0.056687
Turbo Prop       0.048278
Turbo Shaft      0.046596
Turbo Fan        0.015511
Unknown          0.009282
Turbo Jet        0.009095
Electric         0.000125
Hybrid Rocket    0.000062
Name: proportion, dtype: float64
```



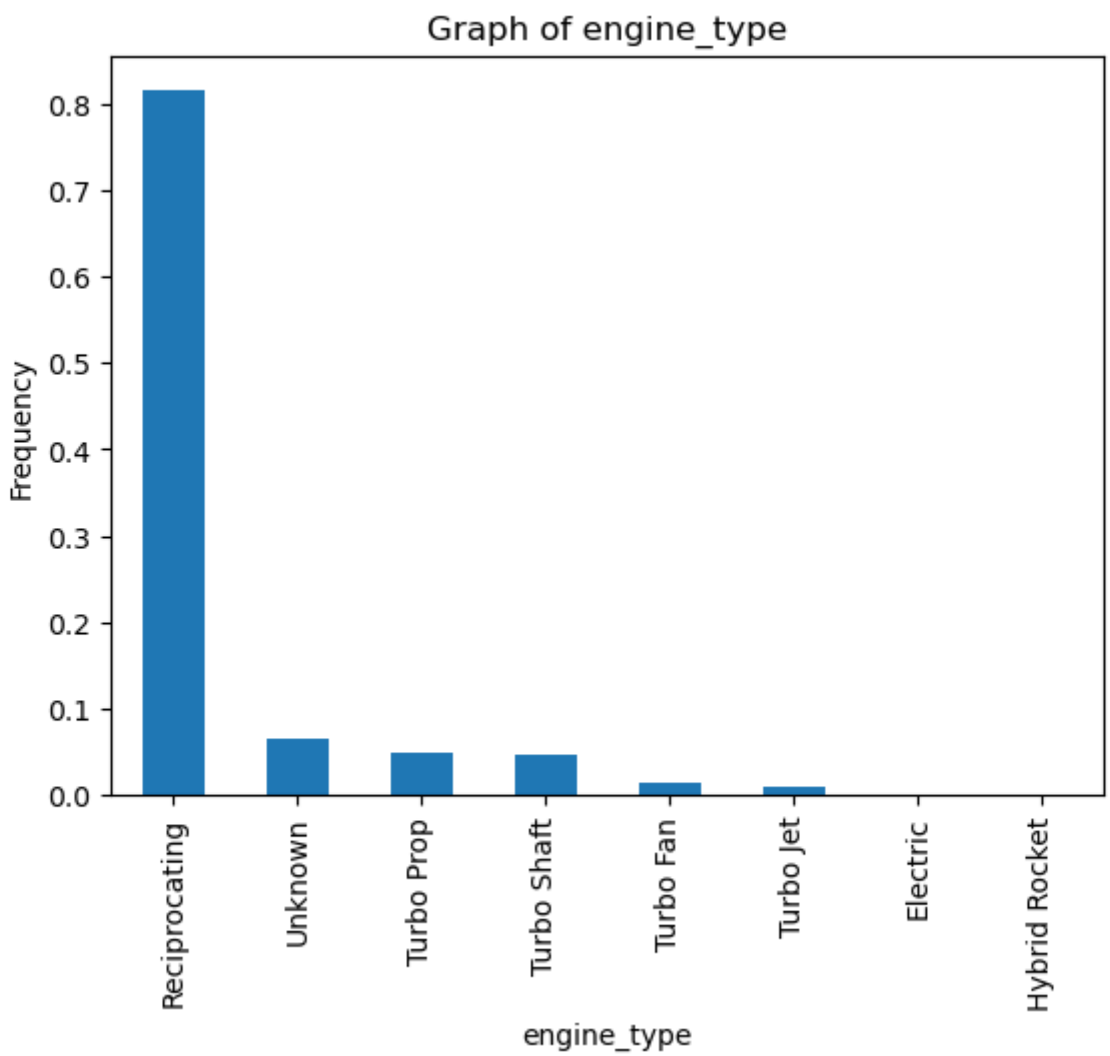
In [74]:
```
# Replace unknown values with 'Unknown' and replace na with 'Unknown'
```

```
df['engine_type'] = df['engine_type'].replace('UNK', 'Unknown')
df['engine_type'].fillna('Unknown', inplace=True)
```

In [75]:

```
plot_column_data(df,'engine_type', 'bar')
```

```
engine_type
Reciprocating    0.814365
Unknown          0.065969
Turbo Prop       0.048278
Turbo Shaft      0.046596
Turbo Fan        0.015511
Turbo Jet        0.009095
Electric         0.000125
Hybrid Rocket    0.000062
Name: proportion, dtype: float64
```



An overwhealming majority of the accidents (ie in 81% of the cases) the engine type of the aircraft was reciprocating

Purpose of Flight

In [76]:

```
df['purpose_of_flight'].value_counts(normalize=True, dropna=False)
```

Out[76]:
```
purpose_of_flight
Personal                    0.649100
Instructional               0.066343
Unknown                     0.063041
Business                    0.058556
Aerial Application          0.031770
NaN                         0.029029
Positioning                 0.025042
Other Work Use              0.017068
Aerial Observation          0.011462
Public Aircraft             0.010901
Ferry                       0.007787
Executive/corporate         0.007413
Flight Test                 0.006292
Skydiving                   0.003800
Air Race/show               0.002741
External Load               0.001931
Air Race show               0.001557
Banner Tow                  0.001308
Public Aircraft - Federal   0.001308
Glider Tow                  0.000997
Public Aircraft - State     0.000872
Public Aircraft - Local     0.000685
Firefighting                0.000623
ASHO                        0.000311
Air Drop                    0.000062
Name: proportion, dtype: float64
```

In [77]:
```
# Let's fill null values with unknown

df['purpose_of_flight'].fillna('Unknown', inplace=True)
```

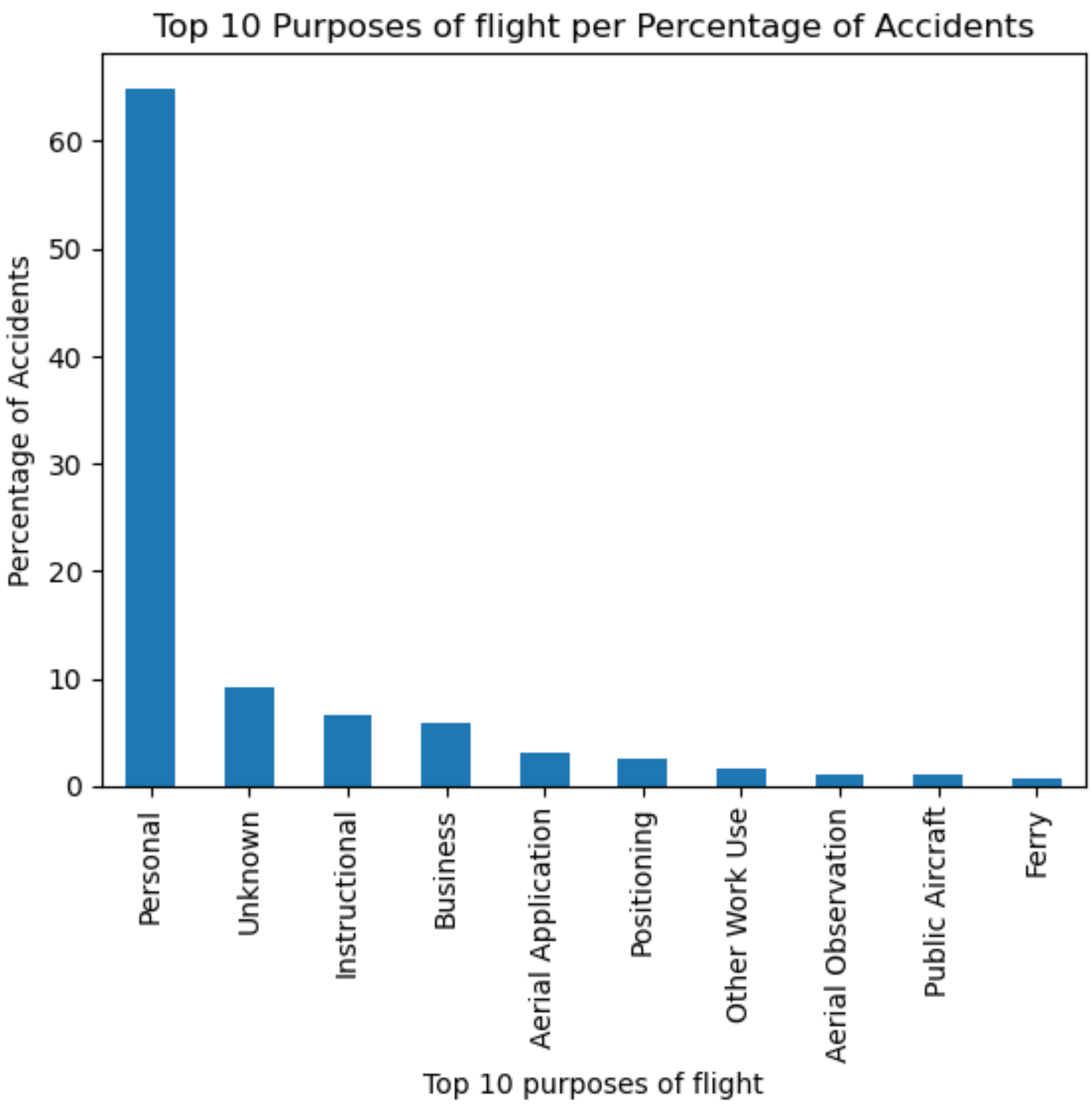
I am going to look at the top 10 of the instances

In [78]:
```
top_10_purpose_of_flight = df['purpose_of_flight'].value_counts(normalize=Tru
top_10_purpose_of_flight
```

Out[78]:
```
purpose_of_flight
Personal              64.909986
Unknown                9.207002
Instructional          6.634274
Business               5.855603
Aerial Application     3.176976
Positioning            2.504205
Other Work Use         1.706846
Aerial Observation     1.146203
Public Aircraft        1.090139
Ferry                  0.778671
Name: proportion, dtype: float64
```

In [79]:
```
plt.figure()
top_10_purpose_of_flight.plot(kind='bar')
plt.xlabel('Top 10 purpose''s of flight')
plt.ylabel('Percentage of Accidents')
```

```
plt.xticks(rotation=90)
plt.title('Top 10 Purpose''s of flight per Percentage of Accidents');
```



The majority of the accidents happened under personal reasons apparently. This doesn't give much insight to our study

Weather condition

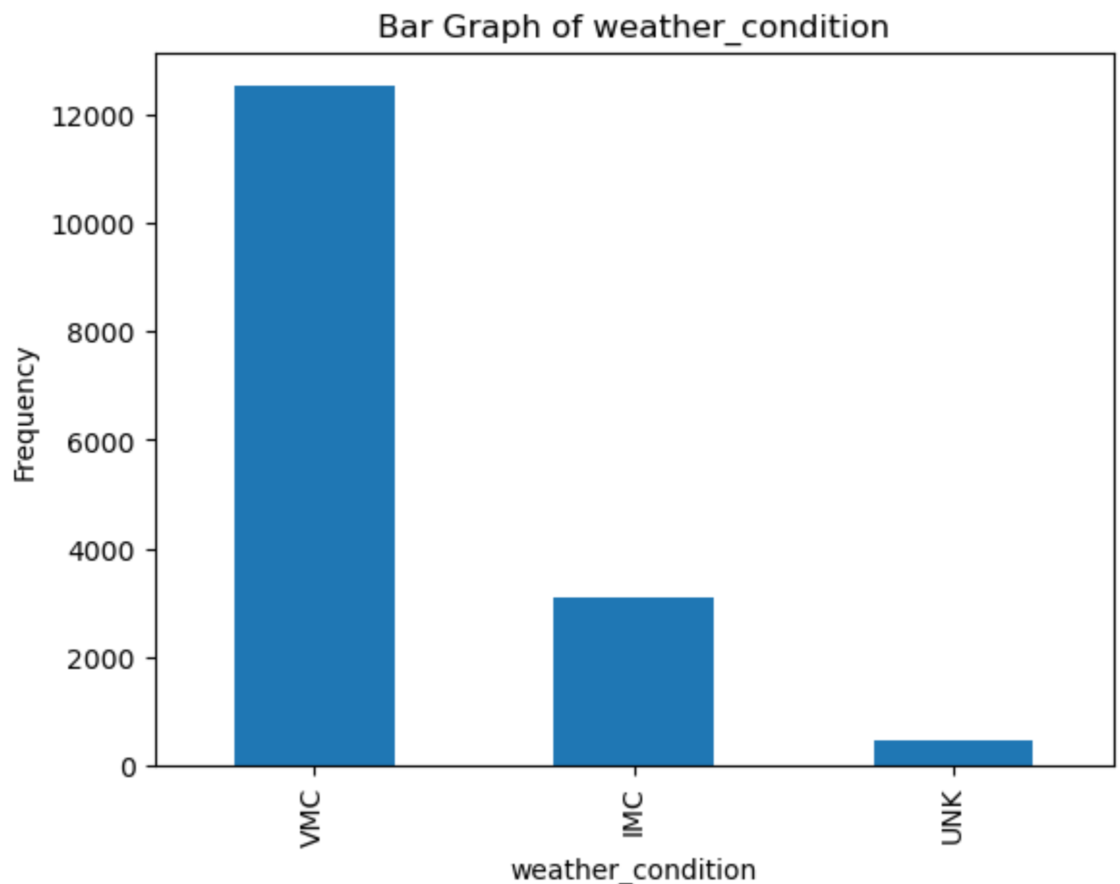
In [80]:
```
df['weather_condition'].value_counts(normalize=True, dropna=False)
```

Out[80]:
```
weather_condition
VMC    0.779044
IMC    0.192550
UNK    0.017006
NaN    0.008534
Unk    0.002866
Name: proportion, dtype: float64
```

In [81]:
```
# Let's replace Unk to UNK and all the null values let's call them: UNK (this
```

```
df['weather_condition'] = df['weather_condition'].replace('Unk', 'UNK')
df['weather_condition'].fillna('UNK', inplace=True)
```

In [82]:
```
plot_bar_graph_for_columns('weather_condition')
```



The dataset contains in its majority VMC weather conditions (ie more than 91% of the data) which says that the flight conditions are good enough for pilots to fly using only visual cues. There is a remaining 7% of flights that were IMC and that are weather conditions that are so poor that pilots cannot safely fly using only visual cues.

Report Status

In [83]:
```
df['report_status'].value_counts(normalize=True, dropna=False)
```

Out[83]:
```
report_status
Probable Cause
0.764219
NaN
0.051019
Factual
0.000934
An in-flight loss of control for undetermined reasons.
0.000249
A loss of control for undetermined reasons.
0.000249
```

```
...
The pilot’s inadvertent pulling of the mixture control lever on takeoff, whic
h shut down the engine.
0.000062
The pilot's loss of airplane control during cruise flight.
0.000062
The pilot’s intentional low-altitude maneuvering and failure to maintain clea
rance from terrain due to distraction.\r
0.000062
The pilot did not maintain adequate airspeed while maneuvering at low altitud
e, which resulted in an aerodynamic stall.
0.000062
The pilot’s failure to secure the magneto switch before attempting to hand ro
tate the engine which resulted in an inadvertent engine start, a runaway airp
lane, and subsequent impact with parked airplanes. Contributing to the accide
nt was the failure to properly secure the airplane with chocks.    0.000062
Name: proportion, Length: 2916, dtype: float64
```

In [84]:

```
# df['report_status'].unique()
```

In [85]:

```
(df['report_status'].isnull().sum()/len(df['report_status']))*100
```

Out[85]:

```
5.101850121472622
```

I'm going to create a new category that groups pilot's faults

In [86]:

```
df['report_status'] = df['report_status'].map(lambda x: 'pilot failure' if is
```

In [87]:

```
# df['report_status'].unique()
```

I'm going to create a new category that groups collisions

In [88]:

```
df['report_status'] = df['report_status'].map(lambda x: 'Collision' if isinst
```

In [89]:

```
df['report_status'].nunique()
```

Out[89]:

```
418
```

There are still 4358 unique cases in the report_status column and the majority of the explanations are not very clear

3.7 Results

As a whole these are the 3 Business recommendations:

1. First business recommendation: Invest in multi-engine aircraft for enhanced safety and reliability.

2. Second business recommendation: Prioritize investment in aircraft with Turbo Shaft, Turbo Prop, or Turbo Jet engines for better performance and efficiency.
3. Third business recommendation: Focus investments on aircraft makers with lower historical injury rates, such as: Beech, Bell, and Boeing.

Here is an in-depth description of the recommendations:

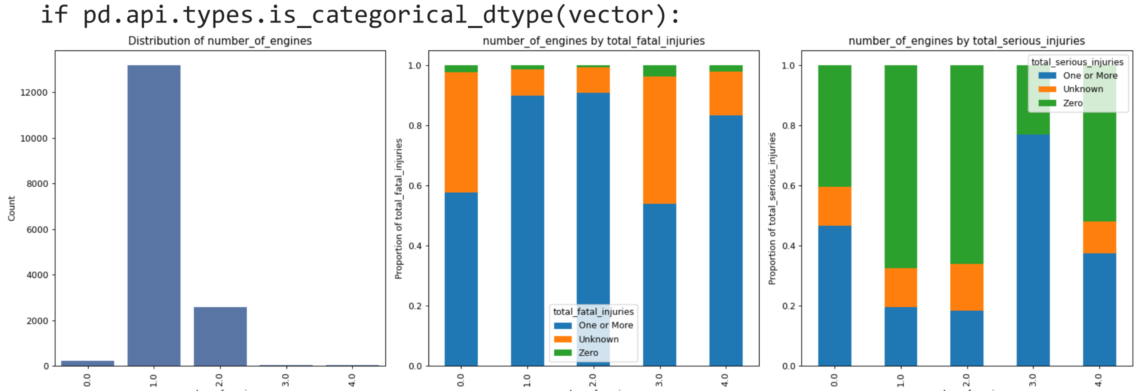
First business recommendation: Invest in aircrafts with more than one engine. The dataset shows that 82% of the accidents happened with aircrafts that had only 1 engine.

This implies that aircraft with more than one engine may have a better safety record and could represent a safer investment.

In [90]:

```
plot_feature(df, 'number_of_engines', 'bar', 'total_fatal_injuries', 'total_s
```

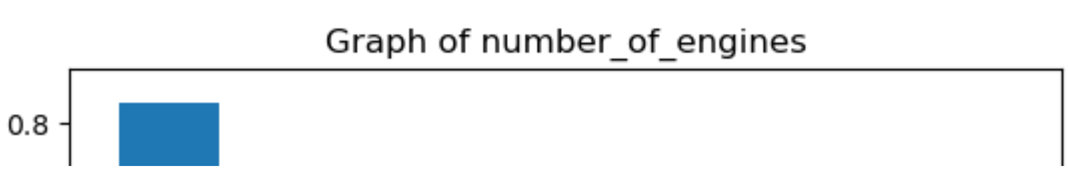
```
C:\Users\Usuario\anaconda3\envs\aircraft_env\Lib\site-packages\seaborn\_oldcor
e.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be remov
ed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\Usuario\anaconda3\envs\aircraft_env\Lib\site-packages\seaborn\_oldcor
e.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be remov
ed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

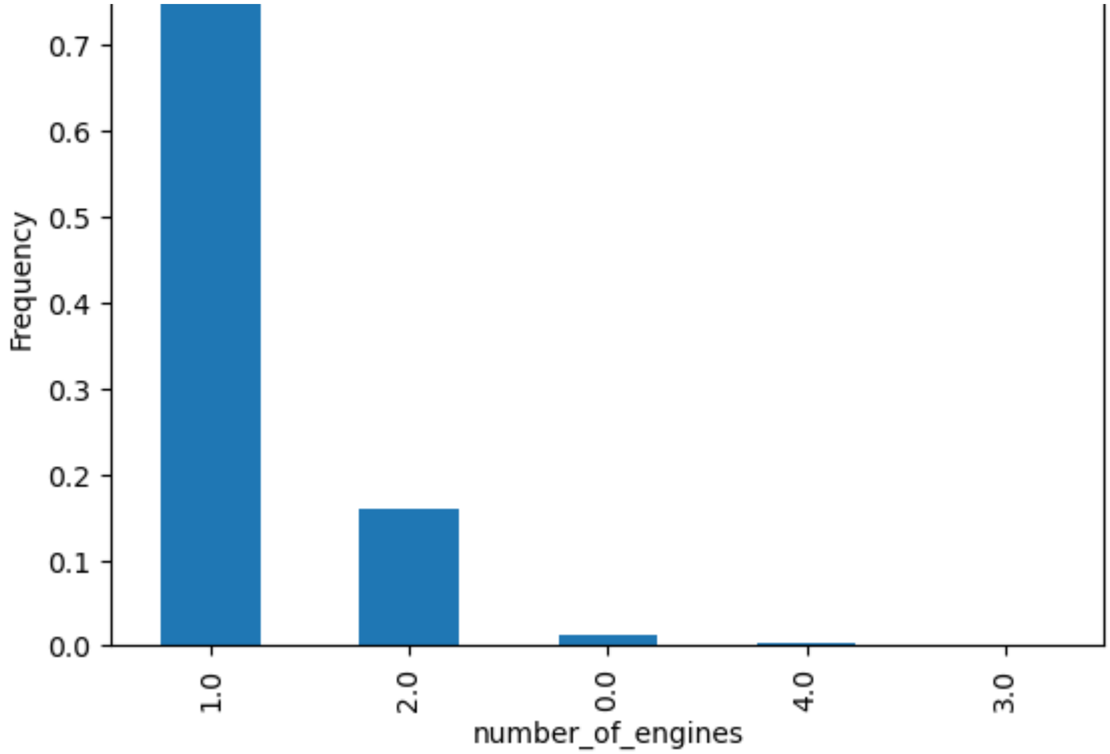


In [91]:

```
plot_column_data(df,'number_of_engines', 'bar')
```

```
number_of_engines
1.0    0.821342
2.0    0.160780
0.0    0.013269
4.0    0.002990
3.0    0.001620
Name: proportion, dtype: float64
```



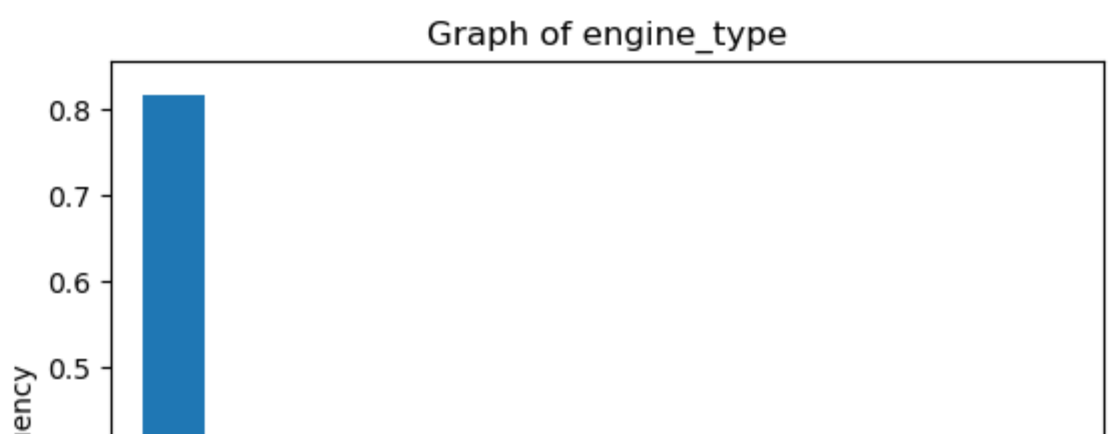


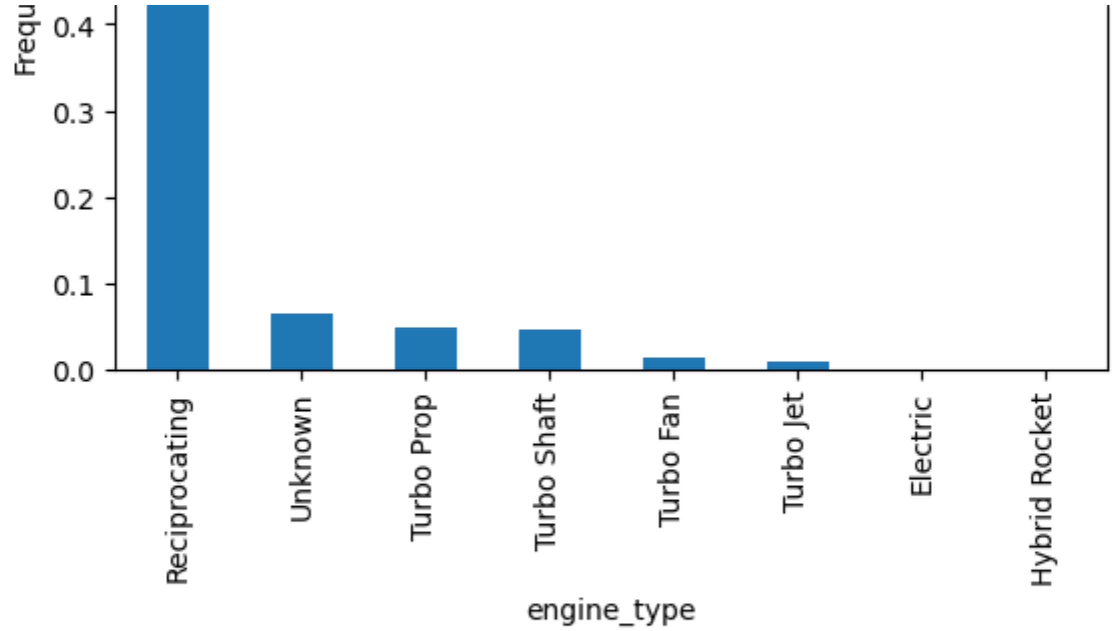
Second business recommendation: Do not invest in aircrafts with a reciprocating engine type. The dataset shows that 81.4% of the accidents happened with aircrafts that had this type of engine.

Investing in aircraft with alternative engine types might reduce risk exposure.

In [92]:
```
plot_column_data(df,'engine_type', 'bar')
```

```
engine_type
Reciprocating    0.814365
Unknown          0.065969
Turbo Prop       0.048278
Turbo Shaft      0.046596
Turbo Fan        0.015511
Turbo Jet        0.009095
Electric         0.000125
Hybrid Rocket    0.000062
Name: proportion, dtype: float64
```

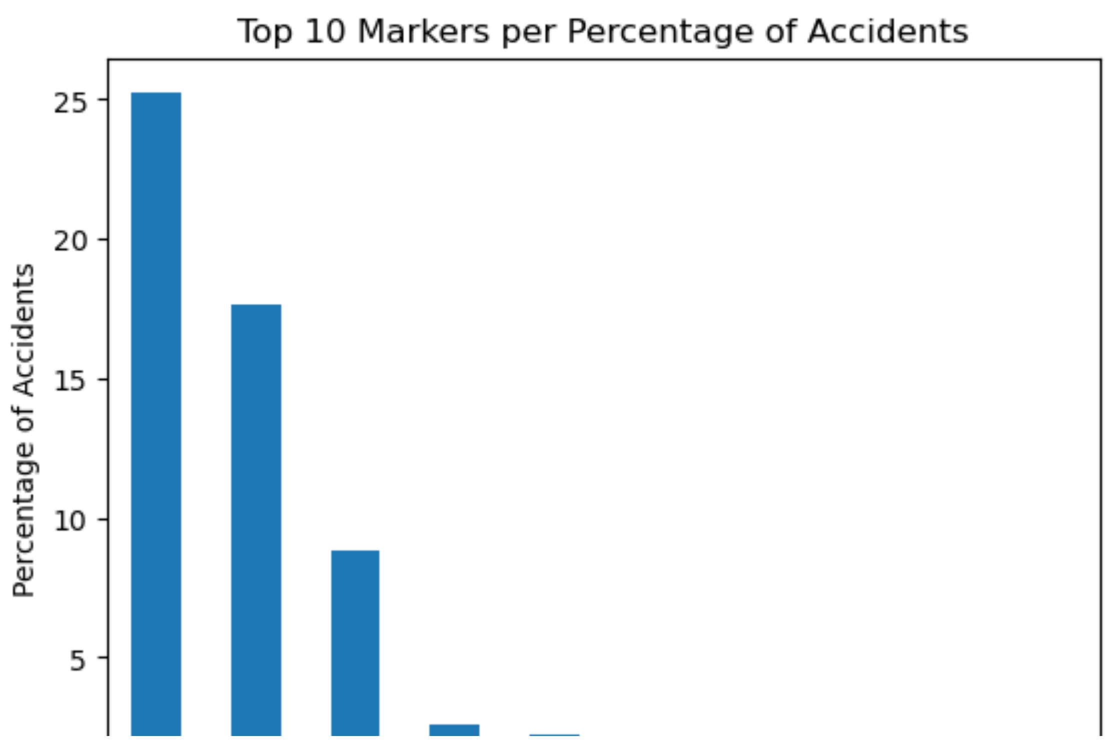


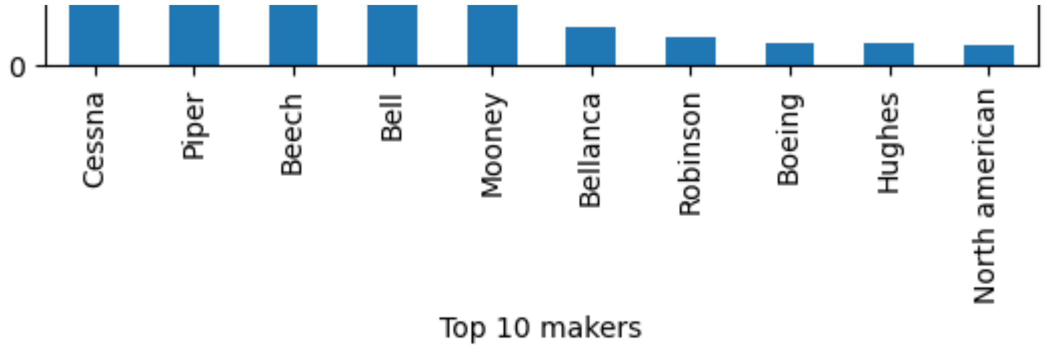


Third business recommendation: Be weary of investing in certain aircraft makers. Be very carefull in investing on 'Cessna', 'Piper' and 'Beech' as their aircrafts combined have had around 50% of the fatal and serious accidents. In particular, 'Cessna' has 25%, 'Piper' 17.7%, and 'Beech' 8.8%. The rest of the makers are involved in less than 2.6% of the fatal and serious accidents.

In [93]:

```
plt.figure()
top_10_make.plot(kind='bar')
plt.xlabel('Top 10 makers')
plt.ylabel('Percentage of Accidents')
plt.xticks(rotation=90)
plt.title('Top 10 Markers per Percentage of Accidents');
```





A prudent investment strategy might be to diversify into manufacturers with lower accident rates or to further scrutinize the causes of these accidents to determine if they are related to maintenance practices, specific models, or flight circumstances that could be mitigated.

Considering the above recommendations, this would be the resulting dataset to consider for aircraft investment