Aircraft analysis

Introduction

In this analysis data of aircraft crashes will be used to analyze the risk associated with types of aircrafts

** commercial aircrafts* ¶

** private aircrafts*

Objective

Find aircrafts with the highest and lowest risk by make, model, category and manufacturer

** which commercial aircraft has the highest and lowest risk*

** which private aircraft has the highest and lowest risk*

** what is the survival rate of passengers on each type of aircraft*

** which manufacturer makes the lowest risk aircraft*

In [ ]:

```

```

Import the pandas and matplotlib module and assign to alias

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import chardet
from rich.console import Console
from rich.table import Table

%matplotlib inline
```

Checking for encoding of csv file

making sure we know the encoding of the file that we will be using so we can make sure we use the .read_csv() correctly

In [2]:
```
# using chardet module to detect the files encoding type
# Read a sample of the file

with open("AviationData.csv", "rb") as file:
    result = chardet.detect(file.read(100000))
    print(result)

#df = pd.read_csv("AviationData.csv", encoding=result['encoding'])
```

```
{'encoding': 'ascii', 'confidence': 1.0, 'language': ''}
```

Encoding of source data

as we can see above that encoding type is 'ascii', but instructing the .read_csv() module to open in ascii returns errors due to columns 6, 7 and 28 having dtypes that are not recognized in ascii. after doing some research, the unknown characters to ascii were known to windows-1252 or cp-1252. low_memory=False to prevent a warning message from appearing. For better control, chunksize flag can be specified

In [3]:
```
aviation_data = pd.read_csv('AviationData.csv', encoding = 'cp1252', low_memory
aviation_data.shape

pd.set_option('display.max_columns', None)
aviation_data.tail(5)
```

Out[3]:

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Lat
88884	20221227106491	Accident	ERA23LA093	2022-12-26	Annapolis, MD	United States	
88885	20221227106494	Accident	ERA23LA095	2022-12-26	Hampton, NH	United States	
88886	20221227106497	Accident	WPR23LA075	2022-12-26	Payson, AZ	United States	341
88887	20221227106498	Accident	WPR23LA076	2022-12-26	Morgan, UT	United States	
88888	20221230106513	Accident	ERA23LA097	2022-12-29	Athens, GA	United States	

Subsetting data

All data in the source file has no use for our objective, therefore only data that can answer our questions will be used. lets also get some info about the data in order to know the columns we will be using

In [4]:
```
# creating subset of aviation_data that are relevant to analyzing safety risk.

data_subset = aviation_data.iloc[:,[0,2,3,4,10,11,12,14,15,17,18,21,23,24,25,2
data_subset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 88889 entries, 0 to 88888
Data columns (total 18 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Event.Id                88889 non-null  object 
 1   Accident.Number         88889 non-null  object 
 2   Event.Date              88889 non-null  object 
 3   Location                88837 non-null  object 
 4   Injury.Severity         87889 non-null  object 
 5   Aircraft.damage         85695 non-null  object 
 6   Aircraft.Category       32287 non-null  object 
 7   Make                    88826 non-null  object 
 8   Model                   88797 non-null  object 
 9   Number.of.Engines       82805 non-null  float64
 10  Engine.Type             81812 non-null  object 
 11  Purpose.of.flight       82697 non-null  object 
 12  Total.Fatal.Injuries    77488 non-null  float64
 13  Total.Serious.Injuries  76379 non-null  float64
 14  Total.Minor.Injuries    76956 non-null  float64
 15  Total.Uninjured         82977 non-null  float64
 16  Weather.Condition       84397 non-null  object 
 17  Publication.Date        75118 non-null  object 
dtypes: float64(5), object(13)
memory usage: 12.2+ MB
```

In [5]:
```
data_subset.head(5)
```

Out[5]:

	Event.Id	Accident.Number	Event.Date	Location	Injury.Severity	Aircraft.damage
0	20001218X45444	SEA87LA080	1948-10-24	MOOSE CREEK, ID	Fatal(2)	Destroyed
1	20001218X45447	LAX94LA336	1962-07-19	BRIDGEPORT, CA	Fatal(4)	Destroyed
2	20061025X01555	NYC07LA005	1974-08-30	Saltville, VA	Fatal(3)	Destroyed
3	20001218X45448	LAX96LA321	1977-06-19	EUREKA, CA	Fatal(2)	Destroyed
4	20041105X01764	CHI79FA064	1979-08-02	Canton, OH	Fatal(1)	Destroyed

Missing Data

There are some numerical columns that has missing data. We want to preserve the balance of overal data, so I would choose to use the mean to fill in missing values

In [6]:
```
data_subset = data_subset.apply(lambda col: col.fillna(col.mean()) if col.dtyp

data_subset.info()

#data_subset.fillna(data_subset.median(), inplace=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 88889 entries, 0 to 88888
Data columns (total 18 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Event.Id                88889 non-null  object 
 1   Accident.Number         88889 non-null  object 
 2   Event.Date              88889 non-null  object 
 3   Location                88837 non-null  object 
 4   Injury.Severity         87889 non-null  object 
 5   Aircraft.damage         85695 non-null  object 
 6   Aircraft.Category       32287 non-null  object 
 7   Make                    88826 non-null  object 
 8   Model                   88797 non-null  object 
 9   Number.of.Engines       88889 non-null  float64
 10  Engine.Type             81812 non-null  object 
 11  Purpose.of.flight       82697 non-null  object 
 12  Total.Fatal.Injuries    88889 non-null  float64
 13  Total.Serious.Injuries  88889 non-null  float64
 14  Total.Minor.Injuries    88889 non-null  float64
 15  Total.Uninjured         88889 non-null  float64
 16  Weather.Condition       84397 non-null  object 
 17  Publication.Date        75118 non-null  object 
dtypes: float64(5), object(13)
memory usage: 12.2+ MB
```

Injuries of all levels

lets add a new column named ' Total.Injuries ' next to all types of injuries to get a general number for all injuries sustained for each observation. We will use this to calculate aircraft passenger survival rate.

In [7]:
```
data_subset.insert(loc=15, column='Total.Injuries', value=data_subset[['Total.

#data_subset.head(2)
data_subset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 88889 entries, 0 to 88888
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Event.Id                88889 non-null  object 
 1   Accident.Number         88889 non-null  object 
 2   Event.Date              88889 non-null  object 
 3   Location                88837 non-null  object 
 4   Injury.Severity         87889 non-null  object 
 5   Aircraft.damage         85695 non-null  object 
 6   Aircraft.Category       32287 non-null  object 
 7   Make                    88826 non-null  object 
 8   Model                   88797 non-null  object 
 9   Number.of.Engines       88889 non-null  float64
 10  Engine.Type             81812 non-null  object 
 11  Purpose.of.flight       82697 non-null  object 
 12  Total.Fatal.Injuries    88889 non-null  float64
 13  Total.Serious.Injuries  88889 non-null  float64
 14  Total.Minor.Injuries    88889 non-null  float64
 15  Total.Injuries          88889 non-null  float64
 16  Total.Uninjured         88889 non-null  float64
 17  Weather.Condition       84397 non-null  object 
 18  Publication.Date        75118 non-null  object 
dtypes: float64(6), object(13)
memory usage: 12.9+ MB
```

In [8]:
```
#lets get rid of NaN from 'Aircraft.Category'. we know that this has to be a co
# 'Unkown' instead
data_subset['Aircraft.Category'] = data_subset['Aircraft.Category'].fillna('Unk
data_subset.head(3)
```

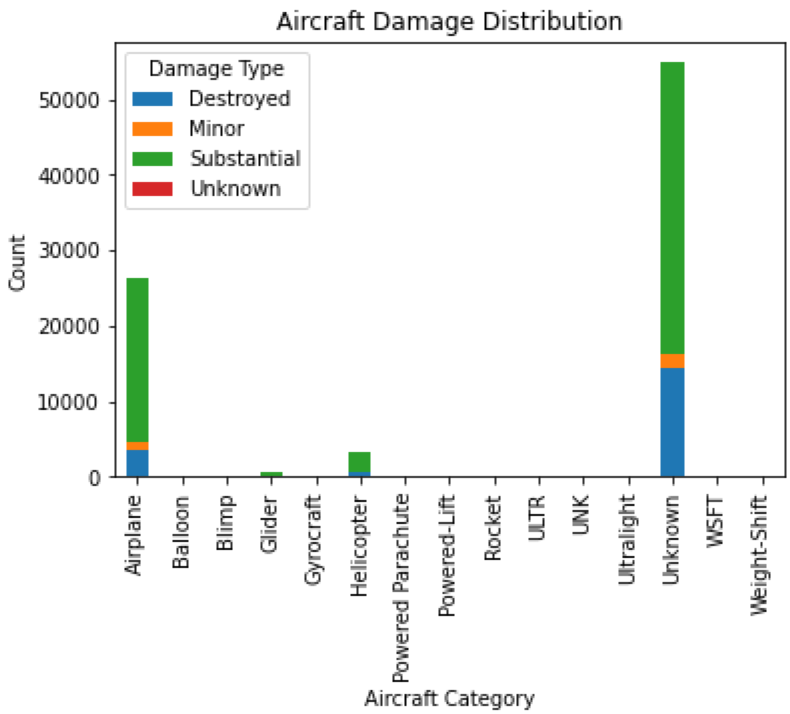
Out[8]:

	Event.Id	Accident.Number	Event.Date	Location	Injury.Severity	Aircraft.damage
0	20001218X45444	SEA87LA080	1948-10-24	MOOSE CREEK, ID	Fatal(2)	Destroyed
1	20001218X45447	LAX94LA336	1962-07-19	BRIDGEPORT, CA	Fatal(4)	Destroyed
2	20061025X01555	NYC07LA005	1974-08-30	Saltville, VA	Fatal(3)	Destroyed

What type of aircrafts has the highest and lowest risk?

Lets group the aircrafts by category then by damage. Damage will be on the y axis and category will be on the x axis. Based on the graph below some aircrafts has no damage history. But lowest to highest risk are: Glider, Helicopter, Airplane, and unkown.

In [9]:
```
damage_counts = data_subset.groupby(['Aircraft.Category', 'Aircraft.damage']).

pivot_table = damage_counts.pivot(index='Aircraft.Category', columns='Aircraft

# Bar chart (uncomment)
#pivot_table.plot(kind='bar')
# Stacked bar chart
pivot_table.plot(kind='bar', stacked=True)
plt.title('Aircraft Damage Distribution')
plt.xlabel('Aircraft Category')
plt.ylabel('Count')
plt.legend(title='Damage Type')
plt.show()
```

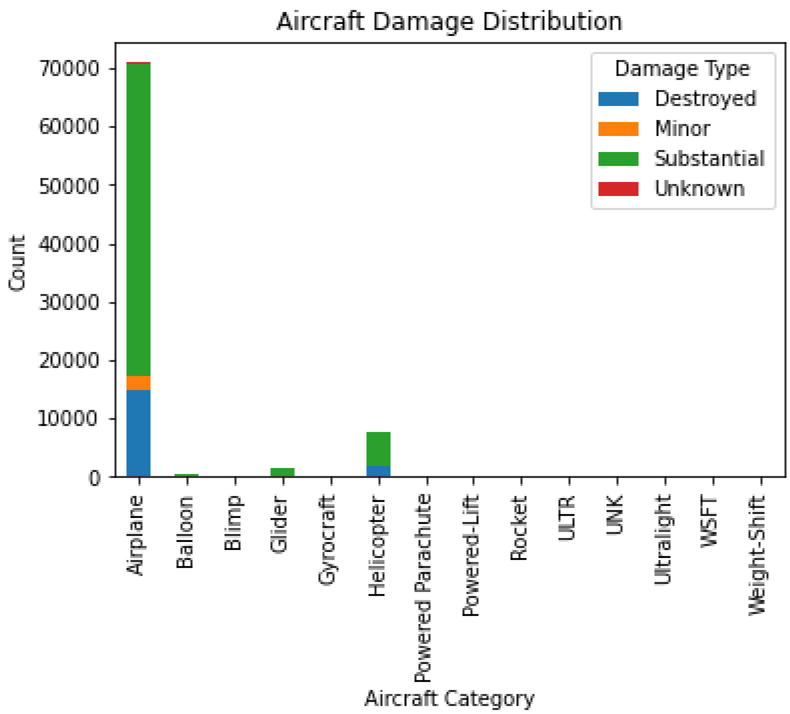


Suprisingly, there are a lot of unknown aircraft in the graph shown above. manually looking at the data, I noticed some entries has 'Unkown' value in the Aircraft.Category. At the same time in the same entry the Make and Model columns has values in them. for example a 'Unkown' value is present for Aircraft.Category column, 'Cessna' value for a Make column and '140' for a Model column is present, but for another entry with the Model being different such as having '180', the Aircraft.Category has a 'unknown' value. I did some research and discovered that a Cessna 140 and 180 looks pretty much the same. I decided to use deductive reasoning to categorize some aircrafts with reference to other columns values.

In [10]:
```
df = data_subset

# Create a mapping from 'Make' & 'Model' to 'Aircraft.Category' where category
category_mapping = (
    df[df['Aircraft.Category'].notna() & (df['Aircraft.Category'] != 'Unknown'
    .groupby(['Make', 'Model'])['Aircraft.Category']
    .agg(lambda x: x.mode()[0] if not x.mode().empty else None)  # Get most fr
    .dropna()
)

# Replace 'Unknown' values by mapping 'Make' & 'Model' combinations
mask = df['Aircraft.Category'].isin(['Unknown', None])  # Identify 'Unknown' r
df.loc[mask, 'Aircraft.Category'] = df.loc[mask, ['Make', 'Model']].apply(
    lambda row: category_mapping.get((row['Make'], row['Model'])), axis=1
)

# Fill remaining 'Unknown' values based only on 'Make' if 'Model' didn't find 
category_mapping_make = (
    df[df['Aircraft.Category'].notna() & (df['Aircraft.Category'] != 'Unknown'
    .groupby('Make')['Aircraft.Category']
    .agg(lambda x: x.mode()[0] if not x.mode().empty else None)
    .dropna()
)

df.loc[mask, 'Aircraft.Category'] = df.loc[mask, 'Make'].map(category_mapping_
```

Now lets graph the values again and see how many more unknown values has been filled with the airplane value in Aircraft.Category column

In [11]:
```
damage_counts = data_subset.groupby(['Aircraft.Category', 'Aircraft.damage']).:

pivot_table = damage_counts.pivot(index='Aircraft.Category', columns='Aircraft

# Bar chart (uncomment)
#pivot_table.plot(kind='bar')
# Stacked bar chart
pivot_table.plot(kind='bar', stacked=True)
plt.title('Aircraft Damage Distribution')
plt.xlabel('Aircraft Category')
plt.ylabel('Count')
plt.legend(title='Damage Type')
plt.show()
```



I wanted to mmake sure that actual airplanes did not have 'Nan' values anymore so I created a dataframe with random samples of data to see if an actual airplane would have the 'Nan' value. if a make and model shows up with a 'Nan' value, I would research the make and model to see if it actually looks like an airplane.

In [12]:
```
df = data_subset
# Filter rows where 'Aircraft.Category' is NaN
nan_category_df = df[df['Aircraft.Category'].isna()]

# Create a new dataset with a random sample of 10 rows (Change n as needed)
nan_category_sample_df = nan_category_df.sample(n=10)

# Display the new dataset
nan_category_sample_df.head(10)
```

Out[12]:

	Event.Id	Accident.Number	Event.Date	Location	Injury.Severity	Aircraft.dan
29718	20001211X13935	ATL92LA036	1992-01-11	CULLMAN, AL	Non-Fatal	Substa
41296	20001208X07466	SEA97LA068	1997-02-24	ARLINGTON, WA	Non-Fatal	Substa
26194	20001212X23721	LAX90DXQ02	1990-07-30	SIERRAVILLE, CA	Non-Fatal	Destr
27461	20001212X16484	MIA91FA085	1991-02-23	POMPANO BEACH, FL	Fatal(1)	Destr
51021	20010727X01537	NYC01LA177	2001-07-16	WALDORF, MD	Non-Fatal	Substa
31594	20001211X15912	MIA93LA003	1992-10-04	HATTIESBURG, MS	Fatal(2)	Substa
53431	20020916X01612	LAX02LA273	2002-09-04	CRESCENT CITY, CA	Fatal(1)	Substa
43178	20001208X09364	NYC98LA044	1997-12-18	OXFORD, CT	Non-Fatal	Substa
58828	20050617X00799	NYC05LA096	2005-06-12	GLOUCESTER, VA	Fatal(1)	Substa
33198	20001211X12883	DEN93FA075	1993-07-08	WALTERS, OK	Fatal(1)	Destr

which aircraft has the highest and lowest risk between private and commercial

the business problem also specifies private and commercial aircrafts. The only column in the data that can be used to determine a private or commercial aircraft is the 'Purpose.of.flight' column. There are several values in this columns that does not specifically point to either category, so some research had to be done in order to categorize every value in this column into a private or commercial value. This step is at my descretion and can skew the data. I may potentially re-classify some of 'Purpose.of.Flight from private to commercial and or the other way around.

Lets create a dictionary that contains the 'Purpose.of.Flight' column values to either Private, Commercial or Unknown

In [13]:
```
# Define the mapping dictionary
purpose_mapping = {
    'Personal': 'Private',
    'Business': 'Private',
    'Instructional': 'Private',
    'Executive/corporate': 'Private',
    'Skydiving': 'Private',
    'Other Work Use': 'Private',
    'Glider Tow': 'Private',
    'Air Race/show': 'Private',

    'Ferry': 'Commercial',
    'Aerial Observation': 'Commercial',
    'Aerial Application': 'Commercial',
    'Public Aircraft': 'Commercial',
    'Public Aircraft - Federal': 'Commercial',
    'Public Aircraft - Local': 'Commercial',
    'Public Aircraft - State': 'Commercial',
    'External Load': 'Commercial',
    'Banner Tow': 'Commercial',
    'Firefighting': 'Commercial',
    'Air Drop': 'Commercial',
    'Positioning': 'Commercial',
    'Flight Test': 'Commercial'
}
```

In [14]:
```
# Creating a new column by mapping values
data_subset['Private_or_Commercial'] = data_subset['Purpose.of.flight'].map(pu

# Filling any unmapped values with 'Unknown'
data_subset['Private_or_Commercial'].fillna('Unknown', inplace=True)

# Would rather this new column is next to the Purpose.of.flight column
data_subset.insert(12, 'Private_or_Commercial', data_subset.pop('Private_or_Co

data_subset.head(2)
```

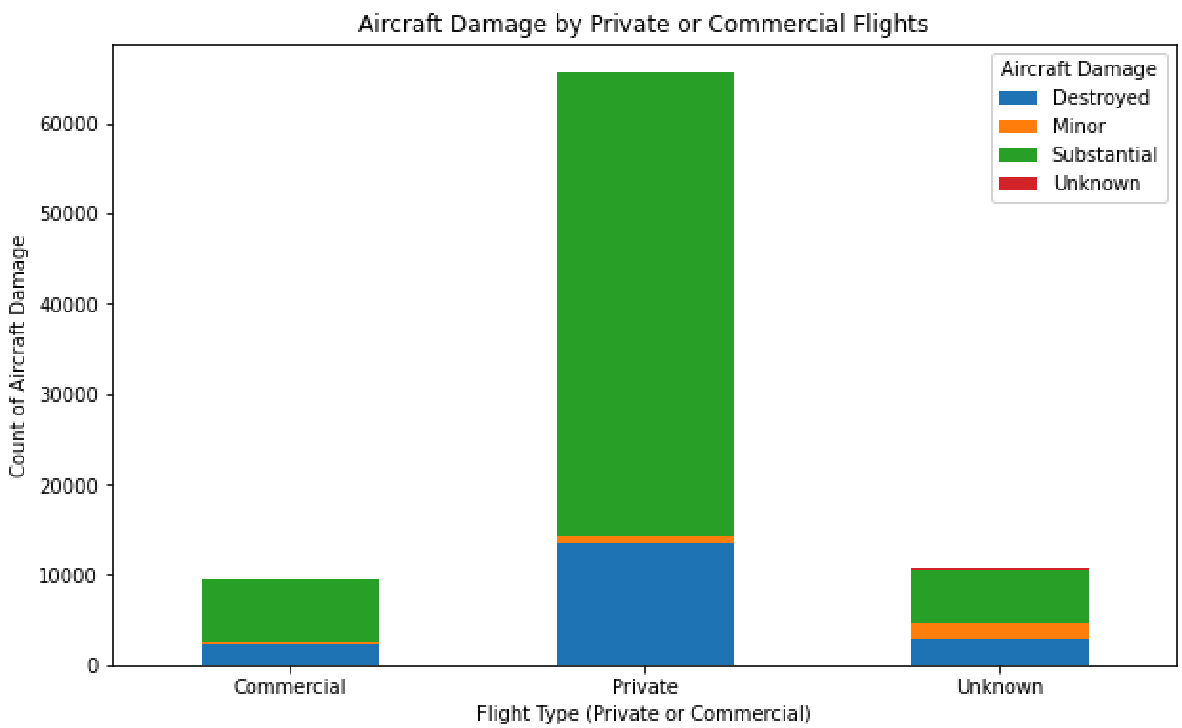
Out[14]:

	Event.Id	Accident.Number	Event.Date	Location	Injury.Severity	Aircraft.damage
0	20001218X45444	SEA87LA080	1948-10-24	MOOSE CREEK, ID	Fatal(2)	Destroyed
1	20001218X45447	LAX94LA336	1962-07-19	BRIDGEPORT, CA	Fatal(4)	Destroyed

Lets group by the Private_or_Commercial and Aircraft.damage in order show a visual representation of the answer to our question (what aircraft has the highest and lowest risk between private and commercial). We can see below that a majority of the aircrafts that got damages are in the private aircraft category.

In [15]:

```
# Count occurrences of each Aircraft.damage type within Private_or_Commercial c
damage_counts = data_subset.groupby(['Private_or_Commercial', 'Aircraft.damage

# Plot the data as a bar chart
damage_counts.plot(kind='bar', figsize=(10, 6), stacked=True)

# Set labels and title
plt.xlabel("Flight Type (Private or Commercial)")
plt.ylabel("Count of Aircraft Damage")
plt.title("Aircraft Damage by Private or Commercial Flights")
plt.xticks(rotation=0)

# Add legend
plt.legend(title="Aircraft Damage")

# Show the plot
plt.show()
```



The data available dates back as early as 1948. The service life of an airplane is 20 to 30 years, so lets focus on only plane 'Event.Date' within the last 20 years. I would even say only last 10 years, but that may drastically decrease our data set. Also based on our intended use of an aircraft, airplanes are the only suitable type of aircraft, so lets create a new dataframe that has only 'airplane' in the 'Aircraft.Category' column

In [16]:
```
ap_after_2004 = data_subset[(data_subset['Aircraft.Category'] == 'Airplane')
                & (data_subset['Event.Date'] >= '2005-01-01')].copy()

# removed decimals from 'Number.of.Engines'
ap_after_2004['Number.of.Engines'] = ap_after_2004['Number.of.Engines'].astype
ap_after_2004.head()
```

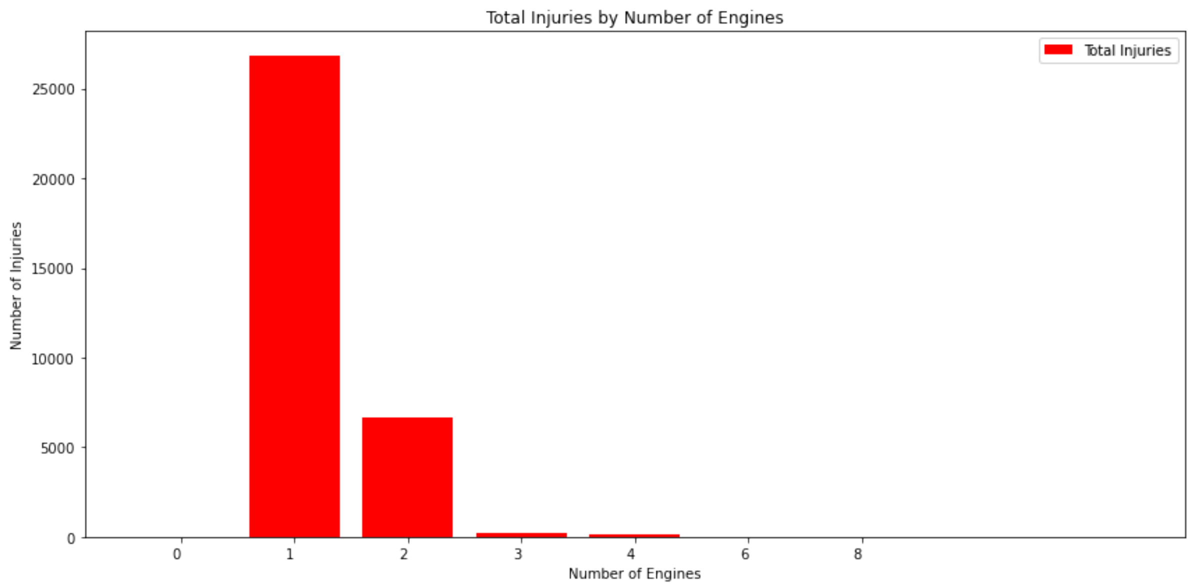
Out[16]:

	Event.Id	Accident.Number	Event.Date	Location	Injury.Severity	Aircraft.d
58015	20050119X00067	IAD05LA030	2005-01-01	FALMOUTH, MA	Non-Fatal	Sub
58016	20050119X00068	IAD05LA029	2005-01-01	POUGHKEEPSIE, NY	Non-Fatal	Sub
58017	20050106X00025	CHI05LA050	2005-01-01	AINSWORTH, NE	Non-Fatal	Sub
58018	20050111X00034	ANC05LA021	2005-01-01	CHICKALOON, AK	Non-Fatal	Sub
58019	20050106X00023	LAX05FA058	2005-01-02	PALO ALTO, CA	Non-Fatal	Sub

which manufacturer makes the lowest risk aircraft

unfortunately the data set does not have a column for manufacturers. Intuitively I attempted to plot 'Total.Injuries' against 'Make', but there are thousands of unique makes in the 'Make' column and does not plot a good graph. I was hoping to get a risk profile based on manufacturer. I decided to substitute 'Make' with 'Number.of.Engines' column

In [17]:
```
# Group by 'Make', summing up injuries
grouped_df = ap_after_2004.groupby(['Number.of.Engines'])[['Total.Injuries']].

# Create figure and axis
fig, ax = plt.subplots(figsize=(12, 6))

# Plot bar chart for Total Injuries
ax.bar(grouped_df['Number.of.Engines'], grouped_df['Total.Injuries'], color='r

# X-axis labels (Make)
ax.set_xticks(range(len(grouped_df)))
ax.set_xticklabels(grouped_df['Number.of.Engines'], ha="right")

# Labels and title
ax.set_xlabel('Number of Engines')
ax.set_ylabel('Number of Injuries')
ax.set_title('Total Injuries by Number of Engines')
ax.legend()

# Show plot
plt.tight_layout()
plt.show()
```



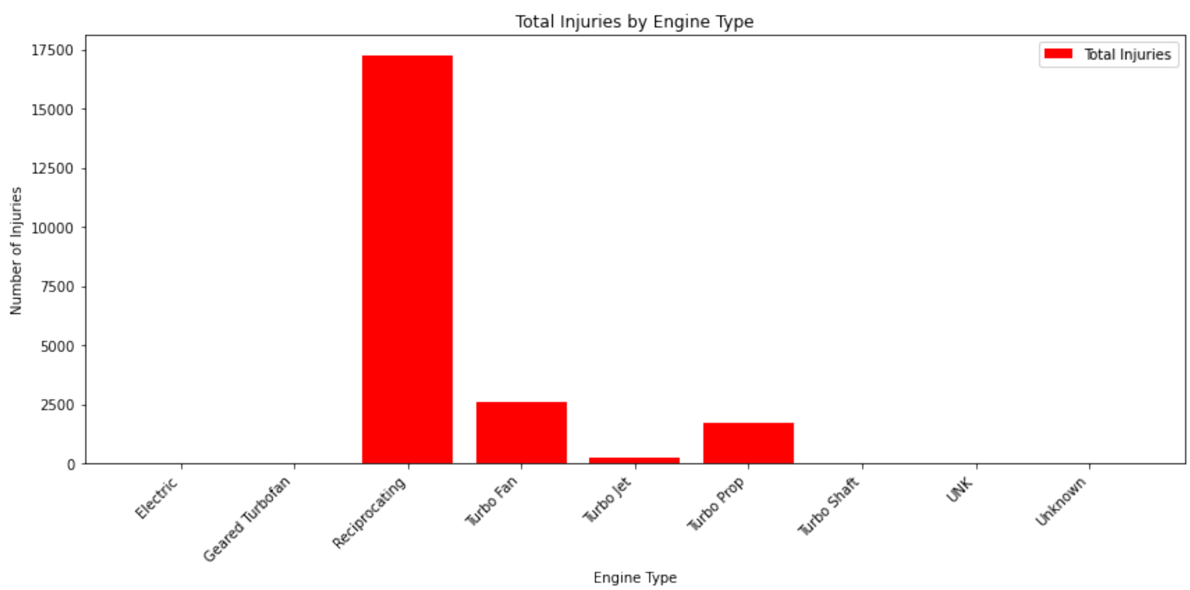
In [18]:
```
grouped_df.head()
```

Out[18]:

	Number.of.Engines	Total.Injuries
0	0	1.284797
1	1	26859.540877
2	2	6682.531329
3	3	224.573005
4	4	129.484487

The code below plots a graph of Total injuries by engine type. Reciprocating engine types by far are the most the most total injuries associated with it and Turbo jet has the least.

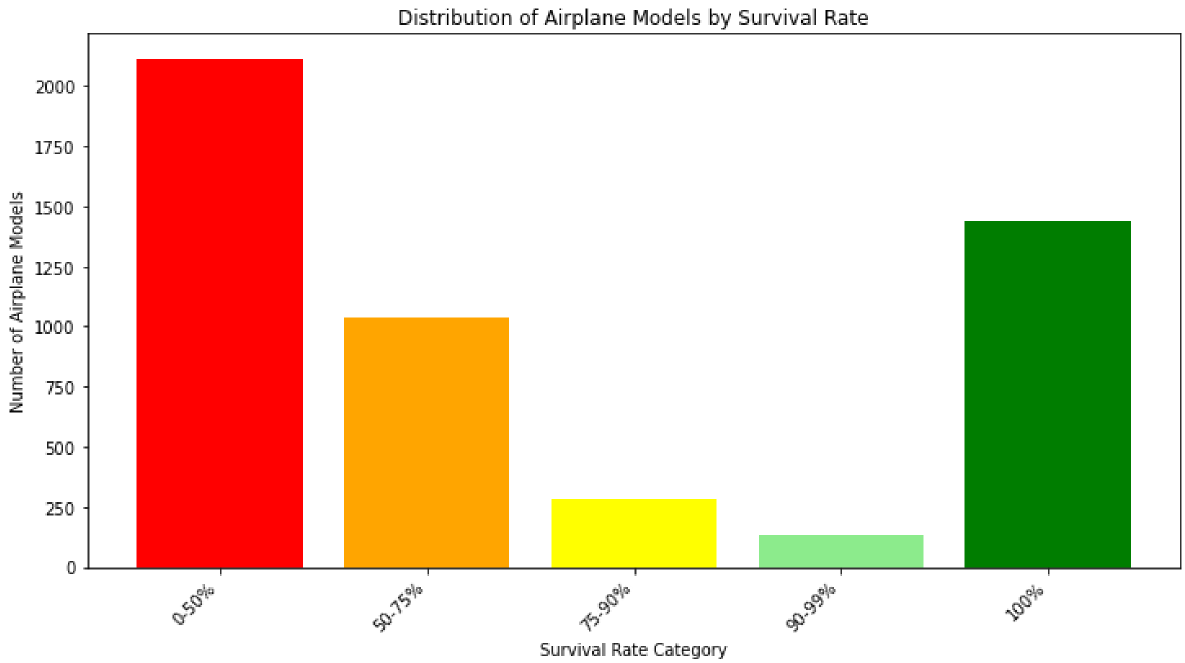
In [19]:
```
grouped_df = ap_after_2004.groupby('Engine.Type')[['Total.Injuries']].sum().re

# Create figure and axis
fig, ax = plt.subplots(figsize=(12, 6))

# Plot bar chart for Total Injuries by Engine Type
bars = ax.bar(grouped_df['Engine.Type'], grouped_df['Total.Injuries'], color='

# Set proper x-ticks and labels
ax.set_xticks(range(len(grouped_df)))  # Set tick positions
ax.set_xticklabels(grouped_df['Engine.Type'], rotation=45, ha="right")  # Set 

# Labels and title
ax.set_xlabel('Engine Type')
ax.set_ylabel('Number of Injuries')
ax.set_title('Total Injuries by Engine Type')
ax.legend()

# Show plot
plt.tight_layout()
plt.show()
```



what is the the survival rate of passengers on each type of airplane

In [20]:
```
import pandas as pd
import matplotlib.pyplot as plt
from rich.console import Console
from rich.table import Table

# Load dataset (replace with actual filename)
df = ap_after_2004

# Filter only 'Airplane' category
df_airplane = df[df['Aircraft.Category'] == 'Airplane'].copy()

# Ensure numerical columns have no NaN values
df_airplane[['Total.Injuries', 'Total.Uninjured']] = df_airplane[['Total.Injur

# Create 'Total.Passengers' column
df_airplane['Total.Passengers'] = df_airplane['Total.Injuries'] + df_airplane[

# Remove rows where Total.Passengers is 0 to avoid division errors
df_airplane = df_airplane[df_airplane['Total.Passengers'] > 0]

# Compute survival rates for each Airplane Model
survival_by_model = df_airplane.groupby('Model')[['Total.Injuries', 'Total.Uni
survival_by_model['Survival Rate'] = survival_by_model['Total.Uninjured'] / su

# Sort models by Survival Rate
survival_by_model = survival_by_model.sort_values('Survival Rate', ascending=Fa

# Define survival rate ranges (bins) and labels
bins = [0, 0.5, 0.75, 0.9, 0.99, 1.0]
labels = ["0-50%", "50-75%", "75-90%", "90-99%", "100%"]
colors = ['red', 'orange', 'yellow', 'lightgreen', 'green']  # Matching colors

# Assign each model to a survival category
survival_by_model['Survival Category'] = pd.cut(survival_by_model['Survival Ra

# Count how many airplane models fall into each survival category
category_counts = survival_by_model['Survival Category'].value_counts().sort_i

# Plot grouped survival rates
fig, ax = plt.subplots(figsize=(12, 6))
bars = ax.bar(category_counts.index, category_counts.values, color=colors)

ax.set_xlabel("Survival Rate Category")
ax.set_ylabel("Number of Airplane Models")
ax.set_title("Distribution of Airplane Models by Survival Rate")
ax.set_xticks(range(len(category_counts)))
ax.set_xticklabels(category_counts.index, rotation=45, ha="right")

plt.show()

# Use Rich to display a table with model names per category, matching colors
console = Console()

table = Table(title="Airplane Models by Survival Rate Category", show_lines=Tru
table.add_column("Survival Category", justify="center", style="bold cyan")
table.add_column("Airplane Models", justify="left")
```

```
# Loop through each category, applying corresponding color
for category, color in zip(labels, colors):
    models_in_category = survival_by_model[survival_by_model['Survival Categor
    model_display = ', '.join(models_in_category[:5]) + ('...' if len(models_i

    # Apply the same color from the bar chart to the Rich text
    table.add_row(f"[{color}]{category}[/{color}]", f"[{color}]{model_display}

console.print(table)
```



Airplane Models by Survival Rate Category

Survival Category	Airplane Models
0-50%	SEA-ERA, EAGLE 540, EA300, Titan Tornado II, SE5-A...
50-75%	Lightning, PIETENPOL AIR CAMPER, PA46R, JR. SR, MD 11F..
75-90%	DC-3T, A 1B, PA 32-260, 525B, AA 5...
90-99%	777 - 236, A319 132, DC-9-82, 777-236ER, 320-200...
100%	HPL 1 HIGH WING PARA, BL, BT13, BT 15, BRISTELL S-LSA...

In [21]:
```
# Load dataset (replace with actual filename)
df = ap_after_2004

# Filter only 'Airplane' category
df_airplane = df[df['Aircraft.Category'] == 'Airplane'].copy()

# Ensure numerical columns have no NaN values
df_airplane[['Total.Injuries', 'Total.Uninjured']] = df_airplane[['Total.Injur

# Create 'Total.Passengers' column
df_airplane['Total.Passengers'] = df_airplane['Total.Injuries'] + df_airplane[

# Remove rows where Total.Passengers is 0 to avoid division errors
df_airplane = df_airplane[df_airplane['Total.Passengers'] > 0]

# Compute survival rates for each Airplane Make (Manufacturer)
survival_by_make = df_airplane.groupby('Make')[['Total.Injuries', 'Total.Uninj
survival_by_make['Survival Rate'] = survival_by_make['Total.Uninjured'] / surv

# Sort makes by Survival Rate
survival_by_make = survival_by_make.sort_values('Survival Rate', ascending=Fal

# Define survival rate ranges (bins) and labels
bins = [0, 0.5, 0.75, 0.9, 0.99, 1.0]
labels = ["0-50%", "50-75%", "75-90%", "90-99%", "100%"]
colors = ['red', 'orange', 'yellow', 'lightgreen', 'green']  # Matching colors

# Assign each Make to a survival category
survival_by_make['Survival Category'] = pd.cut(survival_by_make['Survival Rate

# Count how many airplane manufacturers fall into each survival category
category_counts = survival_by_make['Survival Category'].value_counts().sort_in

# Plot grouped survival rates
fig, ax = plt.subplots(figsize=(12, 6))
bars = ax.bar(category_counts.index, category_counts.values, color=colors)

ax.set_xlabel("Survival Rate Category")
ax.set_ylabel("Number of Airplane Make")
ax.set_title("Distribution of Airplane Make by Survival Rate")
ax.set_xticks(range(len(category_counts)))
ax.set_xticklabels(category_counts.index, rotation=45, ha="right")
plt.show()

# Use Rich to display a table with manufacturer names per category, matching co
console = Console()

table = Table(title="Airplane Make by Survival Rate Category", show_lines=True
table.add_column("Survival Category", justify="center", style="bold cyan")
table.add_column("Airplane Make", justify="left")

# Loop through each category, applying corresponding color
for category, color in zip(labels, colors):
    makes_in_category = survival_by_make[survival_by_make['Survival Category']
    make_display = ', '.join(makes_in_category[:5]) + ('...' if len(makes_in_ca

    # Apply the same color from the bar chart to the Rich text
```

```
    table.add_row(f"[{color}]{category}[/{color}]", f"[{color}]{make_display}[/

console.print(table)
```