Safest Aircraft Analysis

Free Stock photos by Vecteezy (https://www.vecteezy.com/free-photos)

Overview

This project analyzes the safety of various aircrafts. Descriptive analysis of aviation accident data from 1962 to 2023, reported from the National Transportation Safety Board will highlight specific airplane make and models and their reported data on the number of injuries and airplane design features in the accident reports. This analysis can be used by the company to decide which airplane to purchase and operate with the lowest risk in their new business endeavor.

Business Problem

This image is AI-generated

Analyzing the aviation accident dataset can provide valuable insights that can lead to concrete business recommendations in various aspects of the aviation industry. When considering low-risk aircraft types, I analyzed the accident data to identify aircraft types with the highest survival rates. Based on this analysis, which is explained below, I can recommend that the Boeing 777 is the lowest risk airplane. Furthermore, below are three potential airplane design recommendations based on my analysis:

1. Consider purchasing airplanes with Turbo Fan engines
2. Consider purchasing airplanes that have been professionally built. I would not consider amateur builds.
3. Request any incident damage reports and consider airplanes that sustain only minor damage in accidents and incidents.

It's important to note that any business recommendations derived from data analysis should be accompanied by careful consideration of the specific context, regulatory requirements, and limitations of the dataset. Additionally, these recommendations should be subject to ongoing evaluation and refinement based on updated data and emerging industry practices.

Data Understanding

Free Stock photos by Vecteezy (https://www.vecteezy.com/free-photos)

The National Transportation Safety Board provides public data that includes aviation accident data from 1962 to 2023 about civil aviation accidents and selected incidents in the United States and international waters. Each entry has a unique event ID associated with information such as the type of aircraft, date, fatalities, location, and many other factors.

Data Exploration

In [1]:
```
#import the necessary packages
import pandas as pd
import numpy as np

pd.options.display.max_rows =500
pd.options.display.max_columns =500
np.set_printoptions(threshold=np.inf)#no trunkcated lists
```

In [2]:
```
#Import the 'AviationData.csv' file using a proper encoding
aviation_data = pd.read_csv('./Data/AviationData.csv', index_col=0, he
aviation_data.head()
```

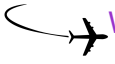
```
/Users/Erin/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
IPython/core/interactiveshell.py:3145: DtypeWarning: Columns (6,7,28)
have mixed types.Specify dtype option on import or set low_memory=Fal
se.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

Out[2]:

Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latit
20001218X45444	Accident	SEA87LA080	1948-10-24	MOOSE CREEK, ID	United States	N
20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPORT, CA	United States	N
20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, VA	United States	36.9
20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, CA	United States	N
20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, OH	United States	N

In [3]:
```
aviation_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 88889 entries, 20001218X45444 to 20221230106513
Data columns (total 30 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Investigation.Type      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   Country                 88663 non-null  object 
 5   Latitude                34382 non-null  object 
 6   Longitude               34373 non-null  object 
 7   Airport.Code            50249 non-null  object 
 8   Airport.Name            52790 non-null  object 
 9   Injury.Severity         87889 non-null  object 
 10  Aircraft.damage         85695 non-null  object 
 11  Aircraft.Category       32287 non-null  object 
 12  Registration.Number     87572 non-null  object 
 13  Make                    88826 non-null  object 
 14  Model                   88797 non-null  object 
 15  Amateur.Built           88787 non-null  object 
 16  Number.of.Engines       82805 non-null  float64
 17  Engine.Type             81812 non-null  object 
 18  FAR.Description         32023 non-null  object 
 19  Schedule                12582 non-null  object 
 20  Purpose.of.flight       82697 non-null  object 
 21  Air.carrier             16648 non-null  object 
 22  Total.Fatal.Injuries    77488 non-null  float64
 23  Total.Serious.Injuries  76379 non-null  float64
 24  Total.Minor.Injuries    76956 non-null  float64
 25  Total.Uninjured         82977 non-null  float64
 26  Weather.Condition       84397 non-null  object 
 27  Broad.phase.of.flight   61724 non-null  object 
 28  Report.Status           82508 non-null  object 
 29  Publication.Date        75118 non-null  object 
dtypes: float64(5), object(25)
memory usage: 21.0+ MB
```



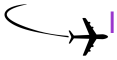
When simply considering the data listed, here's where my thought process starts:

1. Number of columns and the types of variables that can be used
2. Number of Non-Null counts in each column (how complete is the data set/how many rows are there)
3. The type of data and will it need to be standardized

In [4]:
```
#overview/preview of the descriptive statistics
aviation_data.describe(include='all')
```

Out[4]:

	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latitude	Lon
count	88889	88889	88889	88837	88663	34382	
unique	2	88863	14782	27758	219	25592	
top	Accident	WPR23LA045	1984-06-30	ANCHORAGE, AK	United States	332739N	011
freq	85015	2	25	434	82248	19	
mean	NaN	NaN	NaN	NaN	NaN	NaN	
std	NaN	NaN	NaN	NaN	NaN	NaN	
min	NaN	NaN	NaN	NaN	NaN	NaN	
25%	NaN	NaN	NaN	NaN	NaN	NaN	
50%	NaN	NaN	NaN	NaN	NaN	NaN	
75%	NaN	NaN	NaN	NaN	NaN	NaN	
max	NaN	NaN	NaN	NaN	NaN	NaN	



I see that the data set has a lot of missing data, but other than that nothing notable.

In [5]:
```
#copy the data for manipulations
aviation_data_clean = aviation_data.copy()
```

Data Preparation

Data Cleaning

The request from the investors is to recommend an airplane for them to invest. So this reqires knowing a specific make and model. Addressing missing values in the 'Model' and 'Make' columns requires thoughtful consideration, as these columns are crucial for identifying aircrafts involved in accidents.

In [6]:
```
#Clean up the 'Make' and 'Model' columns.
#Create a dictionary to map 'Model' to 'Make'
model_to_make = aviation_data_clean.dropna(subset=['Model', 'Make']).s
```

In [7]:
```
#Create a function to fill missing 'Make' based on 'Model'
def fill_make(row):
    if pd.isnull(row['Make']) and not pd.isnull(row['Model']):
        return model_to_make.get(row['Model'])
    return row['Make']
```

In [8]:
```
#Apply function to fill missing 'Make' values
aviation_data_clean['Make']=aviation_data_clean.apply(fill_make, axis=
```

In [9]:
```
aviation_data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 88889 entries, 20001218X45444 to 20221230106513
Data columns (total 30 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Investigation.Type      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   Country                 88663 non-null  object 
 5   Latitude                34382 non-null  object 
 6   Longitude               34373 non-null  object 
 7   Airport.Code            50249 non-null  object 
 8   Airport.Name            52790 non-null  object 
 9   Injury.Severity         87889 non-null  object 
 10  Aircraft.damage         85695 non-null  object 
 11  Aircraft.Category       32287 non-null  object 
 12  Registration.Number     87572 non-null  object 
 13  Make                    88839 non-null  object 
 14  Model                   88797 non-null  object 
 15  Amateur.Built           88787 non-null  object 
 16  Number.of.Engines       82805 non-null  float64
 17  Engine.Type             81812 non-null  object 
 18  FAR.Description         32023 non-null  object 
 19  Schedule                12582 non-null  object 
 20  Purpose.of.flight       82697 non-null  object 
 21  Air.carrier             16648 non-null  object 
 22  Total.Fatal.Injuries    77488 non-null  float64
 23  Total.Serious.Injuries  76379 non-null  float64
 24  Total.Minor.Injuries    76956 non-null  float64
 25  Total.Uninjured         82977 non-null  float64
 26  Weather.Condition       84397 non-null  object 
 27  Broad.phase.of.flight   61724 non-null  object 
 28  Report.Status           82508 non-null  object 
 29  Publication.Date        75118 non-null  object 
dtypes: float64(5), object(25)
memory usage: 21.0+ MB
```

In this code, I first created a dictionary model_to_make that maps 'Model' values to their corresponding 'Make' values using the non-missing rows in the 'Model' and 'Make' columns.
Then, I defined a function fill_make that takes a row as input. The function checks if the 'Make' value is missing and the 'Model' value is present. If both conditions are met, it uses the model_to_make dictionary to look up the corresponding 'Make' value for that 'Model'.

Finally, I applied the fill_make function to the entire DataFrame using the apply function along axis=1, which means applying the function row-wise.

By running this code, the missing 'Make' values in the DataFrame were filled based on the available 'Model' values in other rows. In this case the 'Model' is more specific and can be used to infer the 'Make' when the value is missing. I thought that this approach would be helpful to fill in some missing values in the 'Make' column and would produce additional values that will contribute to accurately representing this data set. By using this approach, 13 values were inferred and now those rows can be used in the final analysis.

In [10]:
```
#Drop rows with missing values in either 'Make' or 'Model' columns
aviation_data_clean = aviation_data_clean.dropna(subset=['Make', 'Mode
print(aviation_data_clean.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 88790 entries, 20001218X45444 to 20221230106513
Data columns (total 30 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Investigation.Type      88790 non-null  object 
 1   Accident.Number         88790 non-null  object 
 2   Event.Date              88790 non-null  object 
 3   Location                88738 non-null  object 
 4   Country                 88565 non-null  object 
 5   Latitude                34349 non-null  object 
 6   Longitude               34340 non-null  object 
 7   Airport.Code            50216 non-null  object 
 8   Airport.Name            52750 non-null  object 
 9   Injury.Severity         87811 non-null  object 
 10  Aircraft.damage         85617 non-null  object 
 11  Aircraft.Category       32253 non-null  object 
 12  Registration.Number     87511 non-null  object 
 13  Make                    88790 non-null  object 
 14  Model                   88790 non-null  object 
 15  Amateur.Built           88691 non-null  object 
 16  Number.of.Engines       82762 non-null  float64
 17  Engine.Type             81778 non-null  object 
 18  FAR.Description         31945 non-null  object 
 19  Schedule                12534 non-null  object 
 20  Purpose.of.flight       82651 non-null  object 
 21  Air.carrier             16611 non-null  object 
 22  Total.Fatal.Injuries    77403 non-null  float64
 23  Total.Serious.Injuries  76296 non-null  float64
 24  Total.Minor.Injuries    76873 non-null  float64
 25  Total.Uninjured         82889 non-null  float64
 26  Weather.Condition       84349 non-null  object 
 27  Broad.phase.of.flight   61689 non-null  object 
 28  Report.Status           82453 non-null  object 
 29  Publication.Date        75024 non-null  object 
dtypes: float64(5), object(25)
memory usage: 21.0+ MB
None
```

I had to use the dropna() function with the subset parameter set to ['Make', 'Model'], which specifies that I want to drop rows where either the 'Make' or 'Model' column have missing values.
The resulting DataFrame will contain only the rows with both 'Make' and 'Model' values reported. The info() method shows me information about the non-null values in each column to verify that there are no missing values in the 'Make' and 'Model' columns.

After this step, I will work with the cleaned DataFrame, which contains only the rows that have both 'Make' and 'Model' values, without any missing values in those columns.

My next step will be to focus on the rows that contain airplane data since the investors want to purchase an airplane.

Organizing Duplicate Entries

Before anything else can be done to this data it's important to combine like data. For example, the plane make of Cessna is entered as both CESSNA and Cessna for the purposes of value counts, the data entry needs to be standardized.

In [11]:
```
#Convert all data types to string
aviation_data_clean['Model'] = aviation_data_clean['Model'].apply(str)
```

In [12]:
```
#Convert all data to uppercase
aviation_data_clean['Model'] = aviation_data_clean['Model'].str.upper(
```

In [13]:
```
aviation_data_clean['Model'].value_counts()
```

Out[13]:
```
152             2367
172             1756
172N            1168
PA-28-140        932
150              829
                ...
233                1
A600               1
ACRO               1
ERCOUPE415E        1
MONARAI            1
Name: Model, Length: 10838, dtype: int64
```

In [14]:
```
#Convert all data types to string
aviation_data_clean['Make'] = aviation_data_clean['Make'].apply(str)
```

In [15]:
```
#Convert all data to uppercase
aviation_data_clean['Make'] = aviation_data_clean['Make'].map(str.uppe
```

In [16]:
```
aviation_data_clean['Make'].value_counts()
```

Out[16]:
```
CESSNA                    27145
PIPER                     14869
BEECH                      5371
BOEING                     2738
BELL                       2723
                          ...
LYDON PATRICK F               1
GENERAL BALLOON CORP.         1
SANCLEMENTE                   1
HERNANDEZ                     1
BOROM MARCUS P                1
Name: Make, Length: 7575, dtype: int64
```

In [17]:
```
aviation_data_clean.head()
```

Out[17]:

Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latit
20001218X45444	Accident	SEA87LA080	1948-10-24	MOOSE CREEK, ID	United States	N
20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPORT, CA	United States	N
20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, VA	United States	36.9
20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, CA	United States	N
20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, OH	United States	N

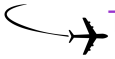
Since the investors have indicated that they are only interested in airplanes, airplane data must be selected from the dataframe.

In [18]:
```
#Identify unique values are listed in the 'Aircraft.Category'
unique_aircraft_values = aviation_data_clean['Aircraft.Category'].unic
unique_aircraft_values
```

Out[18]:
```
array([nan, 'Airplane', 'Helicopter', 'Glider', 'Balloon', 'Gyrocraft
',
       'Ultralight', 'Unknown', 'Blimp', 'Powered-Lift', 'Weight-Shif
t',
       'Powered Parachute', 'Rocket', 'WSFT', 'UNK', 'ULTR'], dtype=o
bject)
```

In [19]:
```
#Use boolean indexing to only look at rows where 'Aircraft.Category' =
aviation_data_clean = aviation_data_clean[aviation_data_clean['Aircraf
aviation_data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 27586 entries, 20170710X52551 to 20221227106497
Data columns (total 30 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Investigation.Type      27586 non-null  object 
 1   Accident.Number         27586 non-null  object 
 2   Event.Date              27586 non-null  object 
 3   Location                27579 non-null  object 
 4   Country                 27579 non-null  object 
 5   Latitude                22082 non-null  object 
 6   Longitude               22073 non-null  object 
 7   Airport.Code            17756 non-null  object 
 8   Airport.Name            18236 non-null  object 
 9   Injury.Severity         26774 non-null  object 
 10  Aircraft.damage         26306 non-null  object 
 11  Aircraft.Category       27586 non-null  object 
 12  Registration.Number     27362 non-null  object 
 13  Make                    27586 non-null  object 
 14  Model                   27586 non-null  object 
 15  Amateur.Built           27569 non-null  object 
 16  Number.of.Engines       24836 non-null  float64
 17  Engine.Type             23373 non-null  object 
 18  FAR.Description         27087 non-null  object 
 19  Schedule                2983 non-null   object 
 20  Purpose.of.flight       23856 non-null  object 
 21  Air.carrier             11257 non-null  object 
 22  Total.Fatal.Injuries    24426 non-null  float64
 23  Total.Serious.Injuries  24369 non-null  float64
 24  Total.Minor.Injuries    24714 non-null  float64
 25  Total.Uninjured         26691 non-null  float64
 26  Weather.Condition       24542 non-null  object 
 27  Broad.phase.of.flight   6390 non-null   object 
 28  Report.Status           22624 non-null  object 
 29  Publication.Date        26587 non-null  object 
dtypes: float64(5), object(25)
memory usage: 6.5+ MB
```



There are 88,790 rows of data that contain both make and model information for aircrafts. Once non airplane data is removed we see in the .info() that the number of rows containing information has dropped to 27,586.

In [20]:
```
aviation_data_clean.to_csv("./Data/aviation_data_clean.csv")
```

Finding Survival Percentage

To find the survival percentage, for each row, I sum fatal injuries, serious injuries, minor injuries, and uninjured to get the total number of passengers. Next, for each unique make/model I sum the columns fatal injuries, serious injuries, minor injuries, uninjured, and total passengers (summing across rows not columns).

My dataset should have only ONE row per unique make/model and this is where I calculate a survival percentage of: 100*((Total.Serious.Injuries + Total.Minor.Injuries + Total.Uninjured) / Total.Passengers.)

I sorted the results by the top 10 airplanes that have a percentage closest to 100%. When I noticed that a number of airplanes had percentages of 100, I sorted additionally by the total number of passengers assuming that more passengers means more profit per flight.

In [21]:
```
#Select only the numeric columns for summing
numeric_columns = aviation_data_clean.select_dtypes(include=[int, floa

#Group by 'Make' and 'Model' and sum the values for each group
aviation_data_combined = aviation_data_clean.groupby(['Make', 'Model']

#Display the combined DataFrame
print(aviation_data_combined)
```

```
                     Make             Model  Number.of.Engines  \
0               177MF LLC       PITTSMODEL12                1.0
1     2007 SAVAGE AIR LLC             EPICLT                1.0
2             2021FX3 LLC          CCX-2000                2.0
3                  3XTRIM          450ULTRA                1.0
4            5 RIVERS LLC              SQ-2                1.0
...                   ...               ...                ...
7265        ZLIN AVIATION       SAVAGECUB-S                1.0
7266               ZODIAC             601XL                0.0
7267        ZUBAIR S KHAN             RAVEN                1.0
7268       ZUBER THOMAS P  ZUBERSUPERDRIFTER                1.0
7269     ZWICKER MURRAY R           GLASTAR                1.0

      Total.Fatal.Injuries  Total.Serious.Injuries  Total.Minor.Injur
ies  \
0                      0.0                     2.0
```

```
0.0
1                       0.0                       0.0
0.0
2                       0.0                       0.0
0.0
3                       1.0                       0.0
1.0
4                       0.0                       0.0
1.0
...                     ...                       ...
...
7265                    0.0                       0.0
0.0
7266                    1.0                       0.0
1.0
7267                    1.0                       0.0
0.0
7268                    0.0                       0.0
0.0
7269                    0.0                       0.0
0.0

      Total.Uninjured
0                 0.0
1                 4.0
2                 4.0
3                 0.0
4                 1.0
...               ...
7265              1.0
7266              0.0
7267              0.0
7268              1.0
7269              2.0

[7270 rows x 7 columns]
```

In [22]:
```
aviation_data_combined[aviation_data_combined['Make'] == 'CESSNA']
```

Out[22]:

	Make	Model	Number.of.Engines	Total.Fatal.Injuries	Total.Serious.Inju
1682	CESSNA	0-1A	1.0	0.0	
1683	CESSNA	120	66.0	5.0	
1684	CESSNA	140	132.0	21.0	
1685	CESSNA	140A	11.0	4.0	
1686	CESSNA	150	238.0	67.0	
1687	CESSNA	150-F	1.0	2.0	
1688	CESSNA	150-G	1.0	0.0	
1689	CESSNA	150-J	1.0	0.0	
1690	CESSNA	150A	4.0	0.0	
1691	CESSNA	150B	3.0	0.0	
1692	CESSNA	150C	7.0	0.0	

In [23]:
```
#Drop Number.of.Engines Column
aviation_data_combined.drop(columns = ['Number.of.Engines'], inplace=T
```

In [24]:
```
aviation_data_combined.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7270 entries, 0 to 7269
Data columns (total 6 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Make                   7270 non-null   object 
 1   Model                  7270 non-null   object 
 2   Total.Fatal.Injuries   7270 non-null   float64
 3   Total.Serious.Injuries 7270 non-null   float64
 4   Total.Minor.Injuries   7270 non-null   float64
 5   Total.Uninjured        7270 non-null   float64
dtypes: float64(4), object(2)
memory usage: 397.6+ KB
```

Looking at the above dataframe, the row count decreased from 27,586 to 7,270 when I grouped all unique make and model sets per row.

In [25]:
```
#Create a total passenger column
#Sum 'Fatal', 'Serious', 'Minor', and 'Uninjured' columns for each row
aviation_data_combined['Total.Passengers'] = aviation_data_combined[['
aviation_data_combined
```

Out[25]:

	Make	Model	Total.Fatal.Injuries	Total.Serious.Injuries	Total.Minor.Inju
0	177MF LLC	PITTSMODEL12	0.0	2.0	
1	2007 SAVAGE AIR LLC	EPICLT	0.0	0.0	
2	2021FX3 LLC	CCX-2000	0.0	0.0	
3	3XTRIM	450ULTRA	1.0	0.0	
4	5 RIVERS LLC	SQ-2	0.0	0.0	
...	...	...	...	...	
7265	ZLIN AVIATION	SAVAGECUB-S	0.0	0.0	
7266	ZODIAC	601XL	1.0	0.0	
7267	ZUBAIR S KHAN	RAVEN	1.0	0.0	
7268	ZUBER THOMAS P	ZUBERSUPERDRIFTER	0.0	0.0	
7269	ZWICKER MURRAY R	GLASTAR	0.0	0.0	

7270 rows × 7 columns

In [26]:
```
#Drop rows with zero total number of passengers; eliminate future NaN
aviation_data_combined = aviation_data_combined[aviation_data_combined
```

Here are a few reasons why the 'Survival.Percentage' might have missing values and why it's important to drop rows with zero total number of passengers at this point:

Division by zero: If any row in the 'Total.Passengers' column has a value of 0, the division (Total.Survivors / Total.Passengers) will result in a NaN value.

Division by NaN: If any row in the 'Total.Passengers' column is NaN, the division will also result in a NaN value.

Missing 'Total.Survivors': If any row has missing values in the 'Total.Serious.Injuries', 'Total.Minor.Injuries', or 'Total.Uninjured' columns, the sum of these columns (Total.Survivors) will be NaN, and the division will result in a NaN value.

In [27]:
```
aviation_data_combined.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7105 entries, 0 to 7269
Data columns (total 7 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Make                   7105 non-null   object 
 1   Model                  7105 non-null   object 
 2   Total.Fatal.Injuries   7105 non-null   float64
 3   Total.Serious.Injuries 7105 non-null   float64
 4   Total.Minor.Injuries   7105 non-null   float64
 5   Total.Uninjured        7105 non-null   float64
 6   Total.Passengers       7105 non-null   float64
dtypes: float64(5), object(2)
memory usage: 444.1+ KB
```

In [28]:
```
aviation_data_combined.to_csv("./Data/aviation_data_combined.csv")
```

Looking at the above dataframe, the row count decreased from 7,270 to 7,105 when I eliminated all of the NaN values.

In [29]:
```

```

```
#Create a survival percentage column
#Sum 'Serious', 'Minor', and 'Uninjured' columns for total survivors i
#Divide total survivors by total passengers for survival rate

aviation_data_combined.loc[:, 'Total.Survivors'] = aviation_data_combi
aviation_data_combined.loc[:, 'Survival.Percentage'] = (aviation_data_

aviation_data_combined


#to surpress the error message
#pd.options.mode.chained_assignment = None  # Suppress the warning

#aviation_data_combined['Total.Survivors'] = aviation_data_combined['T
#aviation_data_combined['Survival.Percentage'] = (aviation_data_combin

#pd.options.mode.chained_assignment = 'warn'  # Restore the warning
```

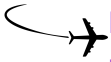
```
/Users/Erin/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
pandas/core/indexing.py:1596: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/panda
s-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
(https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  self.obj[key] = _infer_fill_value(value)
/Users/Erin/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
pandas/core/indexing.py:1745: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/panda
s-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
(https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  isetter(ilocs[0], value)
```

Out[29]:

	Make	Model	Total.Fatal.Injuries	Total.Serious.Injuries	Total.Minor.Inju
0	177MF LLC	PITTSMODEL12	0.0	2.0	
1	2007 SAVAGE AIR LLC	EPICLT	0.0	0.0	
2	2021FX3 LLC	CCX-2000	0.0	0.0	

3	3XTRIM	450ULTRA	1.0	0.0
4	5 RIVERS LLC	SQ-2	0.0	0.0
...	...	...	...	...
7265	ZLIN AVIATION	SAVAGECUB-S	0.0	0.0
7266	ZODIAC	601XL	1.0	0.0
7267	ZUBAIR S KHAN	RAVEN	1.0	0.0
7268	ZUBER THOMAS P	ZUBERSUPERDRIFTER	0.0	0.0
7269	ZWICKER MURRAY R	GLASTAR	0.0	0.0

7105 rows × 9 columns



It looks like there are many airplanes with 100% survival so to narrow it down a bit more, I'm going to also sort by the total passengers column so that we can see both 100% survial of the most people.

In [30]:
```
#Sort the DataFrame based on 'Survival.Percentage' and 'Total.Passenge
sorted_counts = aviation_data_combined.sort_values(by=['Survival.Perce

#Get the top 10 make/model pairs with the most passengers and best sur
top_10_highest_survival_rate = sorted_counts.head(10)


#View results
top_10_highest_survival_rate
```

Out[30]:

	Make	Model	Total.Fatal.Injuries	Total.Serious.Injuries	Total.Minor.Injuries	Total.U
1273	BOEING	777	0.0	6.0	22.0	
1220	BOEING	757	0.0	9.0	4.0	
1197	BOEING	747-400	0.0	0.0	1.0	
1293	BOEING	787	0.0	5.0	3.0	
1279	BOEING	777-222	0.0	5.0	6.0	
1150	BOEING	737-7H4	0.0	5.0	5.0	
1339	BOEING	B777	0.0	0.0	8.0	
304	AIRBUS	A380	0.0	0.0	1.0	
1297	BOEING	787-9	0.0	1.0	2.0	
4618	MCDONNELL DOUGLAS	MD-11	0.0	0.0	200.0	

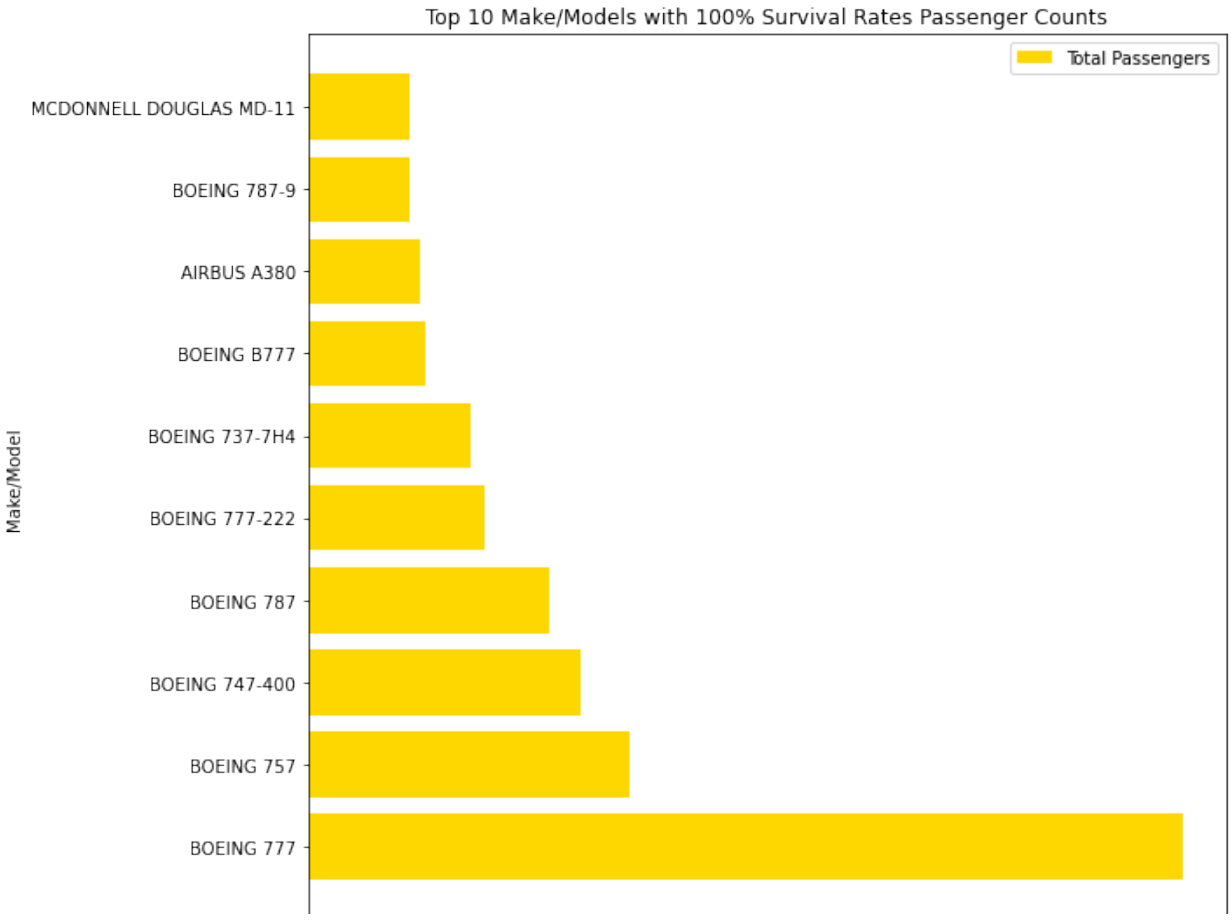
In [31]:
```
aviation_data_combined['Survival.Percentage'].value_counts()
```

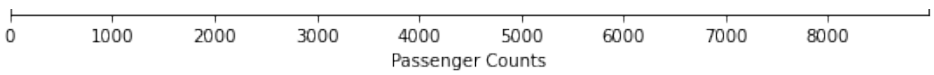
Out[31]:
```
100.000000    4892
0.000000      1056
50.000000      190
66.666667       95
33.333333       46
75.000000       42
80.000000       30
85.714286       24
83.333333       23
60.000000       21
71.428571       20
84.615385       17
77.777778       16
25.000000       14
90.000000       14
40.000000       13
87.500000       12
57.142857       11
86.666667       10
```

The value counts here are giving me a sense of the spread of the survival percentages. It's helpful to see that 4892/7105 (about 70%) of the values for the passenger survival percentage are 100% survival. About 15% of the values have a 0% survival rate and the rest of the 15% of the counts are somewhere in between.

In [32]:
```
```

```
import matplotlib.pyplot as plt

#Sort the DataFrame based on 'Survival.Percentage' in ascending order
sorted_counts = top_10_highest_survival_rate.sort_values(by='Survival.

#Set the figure size for the horizontal bar chart
plt.figure(figsize=(10, 8))

#Plot the horizontal bars for 'Survival Rate'
#plt.barh(sorted_counts['Make'] + ' ' + sorted_counts['Model'], sorted

#Plot the horizontal bars for 'Total Passengers'
plt.barh(sorted_counts['Make'] + ' ' + sorted_counts['Model'], sorted_

#Set labels and title
plt.xlabel('Passenger Counts')
plt.ylabel('Make/Model')
plt.title('Top 10 Make/Models with 100% Survival Rates Passenger Count
plt.legend()

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





This visualization shows that the Boeing 777, by far, has the safest flights with the most people transported. I chose to include the passenger count as a measure of the potential revenue. This assumes that more people occupying seats leads to greater profit for the company.

Further analysis specifically on the Boeing 777 airplane model would provide a more detailed analysis of the risk for that particular airplane.

In [33]:
```
#Locate all Boeing 777 data
boeing_777_data = aviation_data_clean.loc[(aviation_data_clean['Make']

#Display the row for BOEING 777 across all columns
boeing_777_data
```

Out[33]:

Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	La
20001211X10549	Incident	DCA98WA074	1998-07-29	LONDON HEATHROW, United Kingdom	United Kingdom	
20060111X00048	Accident	DCA04MA061	2004-07-25	Miami, FL	United States	
20060425X00482	Accident	DCA06WA035	2006-04-19	Shanghai, China	China	
20081201X44308	Incident	DCA09IA014	2008-11-26	Bozeman, MT	United States	45
20090105X01026	Incident	ENG09IA002	2009-01-	Atlanta, GA	United	03

Recommendation 1:

In this design exploration, I want to understand the survival rate around amateur built airplanes and will be determining the average survival percentage of amateur built airplanes.

When considering whether or not to invest, should we exclude all amateur builds? Are they more dangerous?

In [34]:
```
#View unique values in 'Amateur.Built'
unique_aircraft_builds = aviation_data_clean['Amateur.Built'].unique()
unique_aircraft_builds
```

Out[34]:
```
array(['No', 'Yes', nan], dtype=object)
```

During my previous determination of the plane with the best survival rate, I used aggregated data to calculate the survival rate. If I were to do that in this scenario, I could be elimitating data because there may be make/model pairs that are both professionally manufactured and also amateur built so I will investigate.

In [35]:
```
#Group the data by 'Make' and 'Model' and count the occurrences of dif
make_model_amateur_counts = aviation_data_clean.groupby(['Make', 'Mode

#Filter for combinations of 'Make' and 'Model' with both 'Yes' and 'No
make_model_with_both = make_model_amateur_counts.index[
    make_model_amateur_counts.index.get_level_values('Amateur.Built').
].unique()

#Print the combinations of 'Make' and 'Model' with both 'Yes' and 'No'
print(make_model_with_both)
```

```
MultiIndex([(            '177MF LLC',      'PITTSMODEL12',  'No'),
            (  '2007 SAVAGE AIR LLC',           'EPICLT', 'Yes'),
            (          '2021FX3 LLC',         'CCX-2000',  'No'),
            (          '2021FX3 LLC',         'CCX-2000', 'Yes'),
            (              '3XTRIM',         '450ULTRA',  'No'),
            (         '5 RIVERS LLC',             'SQ-2', 'Yes'),
            (                '737',              '800',  'No'),
            (                '777',              'FF2',  'No'),
            (          '781569 INC',            'FX210', 'Yes'),
            (     'AAA AIRCRAFT LLC',         'CCX-2000',  'No'),
            ...
            ('ZIVKO AERONAUTICS INC',         'EDGE540T',  'No'),
            (                'ZLIN',           'SAVAGE',  'No'),
            (                'ZLIN',             'Z143',  'No'),
            (                'ZLIN',            'Z242L',  'No'),
            (                'ZLIN',              'Z50',  'No'),
            (        'ZLIN AVIATION',      'SAVAGECUB-S',  'No'),
            (              'ZODIAC',             '601XL',  'No'),
            (        'ZUBAIR S KHAN',            'RAVEN', 'Yes'),
            (       'ZUBER THOMAS P', 'ZUBERSUPERDRIFTER',  'No'),
            (     'ZWICKER MURRAY R',           'GLASTAR', 'Yes')],
           names=['Make', 'Model', 'Amateur.Built'], length=7363)
```

It looks like there are combinations of make and model that do have both yes and no values for the amateur build. In this case, I will not be able to use my aggregated survival rate and will recalculate it below based on the cleaned, but not aggregated data.

In [36]:
```
#Select only numeric columns for summing
passenger_injury_columns = aviation_data_clean.select_dtypes(include=[

#Drop rows with zero number of passengers; eliminate future NaN values
aviation_data_built = aviation_data_clean[aviation_data_clean['Total.F

#Group by 'Amateur.Built' and sum the values for each group
aviation_data_built = aviation_data_built.groupby(['Amateur.Built'], a

#Display the combined DataFrame
print(aviation_data_built)
```

```
  Amateur.Built  Number.of.Engines  Total.Fatal.Injuries  \
0            No            21970.0               12737.0
1           Yes             2616.0                 951.0

   Total.Serious.Injuries  Total.Minor.Injuries  Total.Uninjured
0                  6222.0                4176.0         143596.0
1                   650.0                 568.0           1865.0
```

In [37]:
```
#Create a total passenger column
#Sum 'Fatal', 'Serious', 'Minor', and 'Uninjured' columns for each row
aviation_data_built['Total.Passengers'] = aviation_data_built[['Total.
aviation_data_built
```

Out[37]:

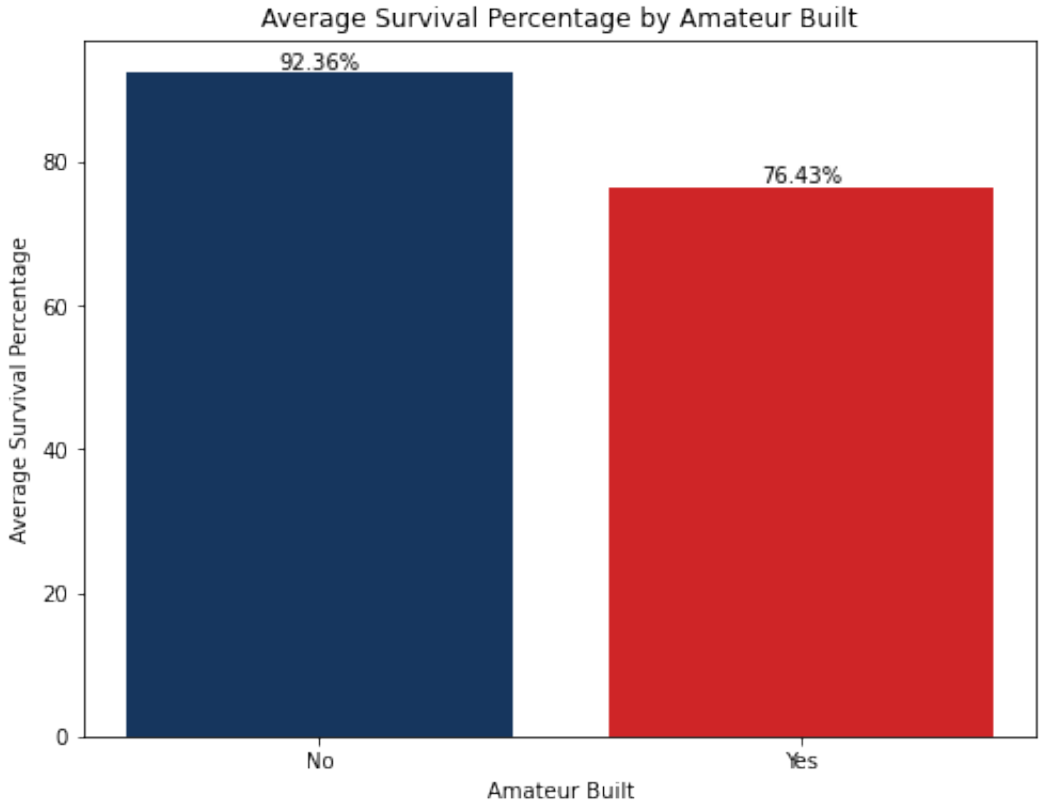
	Amateur.Built	Number.of.Engines	Total.Fatal.Injuries	Total.Serious.Injuries	Total.Minor.Injuries
0	No	21970.0	12737.0	6222.0	4176.0
1	Yes	2616.0	951.0	650.0	568.0

In [38]:
```
#Create a survival percentage column
aviation_data_built.loc[:, 'Total.Survivors'] = aviation_data_built['T
aviation_data_built.loc[:, 'Survival.Percentage'] = (aviation_data_bui

aviation_data_built
```

Out[38]:

	Amateur.Built	Number.of.Engines	Total.Fatal.Injuries	Total.Serious.Injuries	Total.Minor.Injuries
0	No	21970.0	12737.0	6222.0	4176.0
1	Yes	2616.0	951.0	650.0	568.0

In [39]:
```
import matplotlib.pyplot as plt

#Filter the dataframe to include only 'Yes' and 'No' values in the 'Am
filtered_data = aviation_data_built[aviation_data_built['Amateur.Built

#Group by 'Amateur.Built' and calculate the average survival percentag
avg_survival_by_amateur_built = filtered_data.groupby('Amateur.Built')

#Create a bar chart for the average survival percentage for each categ
plt.figure(figsize=(8, 6))
bar_plot = plt.bar(avg_survival_by_amateur_built.index, avg_survival_b

#Set axis labels and title
plt.xlabel('Amateur Built')
plt.ylabel('Average Survival Percentage')
plt.title('Average Survival Percentage by Amateur Built')

#Annotate each bar with its corresponding value (survival percentage)
for bar in bar_plot:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height, f'{height:.2f}

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



This graph show a more than 15% increase in survival when flying on a professionally manufactured airplane. Could potentially explore, percentages within different Makes of airplanes.

Recommendation 2:

In this design exploration, I would like to understand the relationship between the survival rate and the number of engines for each make and model of airplane so I will determine the average survival percentage by engine type.

When considering this investment, should certain types of engines be excluded? Are some more likely to have lower survival percentages?

In [40]:
```
#View unique values listed in 'Engine.Type'
unique_aircraft_engines = aviation_data_clean['Engine.Type'].unique()
unique_aircraft_engines
```

Out[40]:
```
array(['Turbo Fan', 'Reciprocating', 'Turbo Prop', 'Turbo Jet', nan,
       'Unknown', 'Turbo Shaft', 'Electric', 'Geared Turbofan', 'UNK'
],
      dtype=object)
```

In [41]:
```
#Exclude specific values from the 'Engine.Type' column
excluded_values = ['UNK', 'Unknown', np.nan]
filtered_data_engine = aviation_data_clean[~aviation_data_clean['Engir
```

After filtering out any unknown or missing data, I will again have to calculate the passenger survival percentage on the clean, disaggregated data.

In [42]:
```
#Group the data by 'Make' and 'Model' and count the occurrences of dif
#make_model_engine_counts = filtered_data_engine.groupby(['Make', 'Mod

#Filter for combinations of 'Make' and 'Model' by Engine type values
#make_model_by_engine = make_model_engine_counts.index[
#    make_model_engine_counts.index.get_level_values('Engine.Type').is
#.unique()]

#Print the combinations of 'Make' and 'Model' by Engine type values
#print(make_model_by_engine)
```

In [43]:
```

```

```
#Select only numeric columns for summing
passenger_injury_columns = filtered_data_engine.select_dtypes(include=

#Drop rows with zero number of passengers; eliminate future NaN values
aviation_data_engine = filtered_data_engine[filtered_data_engine['Tota

#Group by 'Engine.Type' and sum the values for each group
aviation_data_engine = aviation_data_engine.groupby(['Engine.Type'], a

#Display the combined DataFrame
print(aviation_data_engine)

#Create a total passenger column
# Sum 'Fatal', 'Serious', 'Minor', and 'Uninjured' columns for each ro
aviation_data_engine['Total.Passengers'] = aviation_data_engine[['Tota
aviation_data_engine

#Create a survival percentage column
aviation_data_engine.loc[:, 'Total.Survivors'] = aviation_data_engine[
aviation_data_engine.loc[:, 'Survival.Percentage'] = (aviation_data_en

aviation_data_engine
```

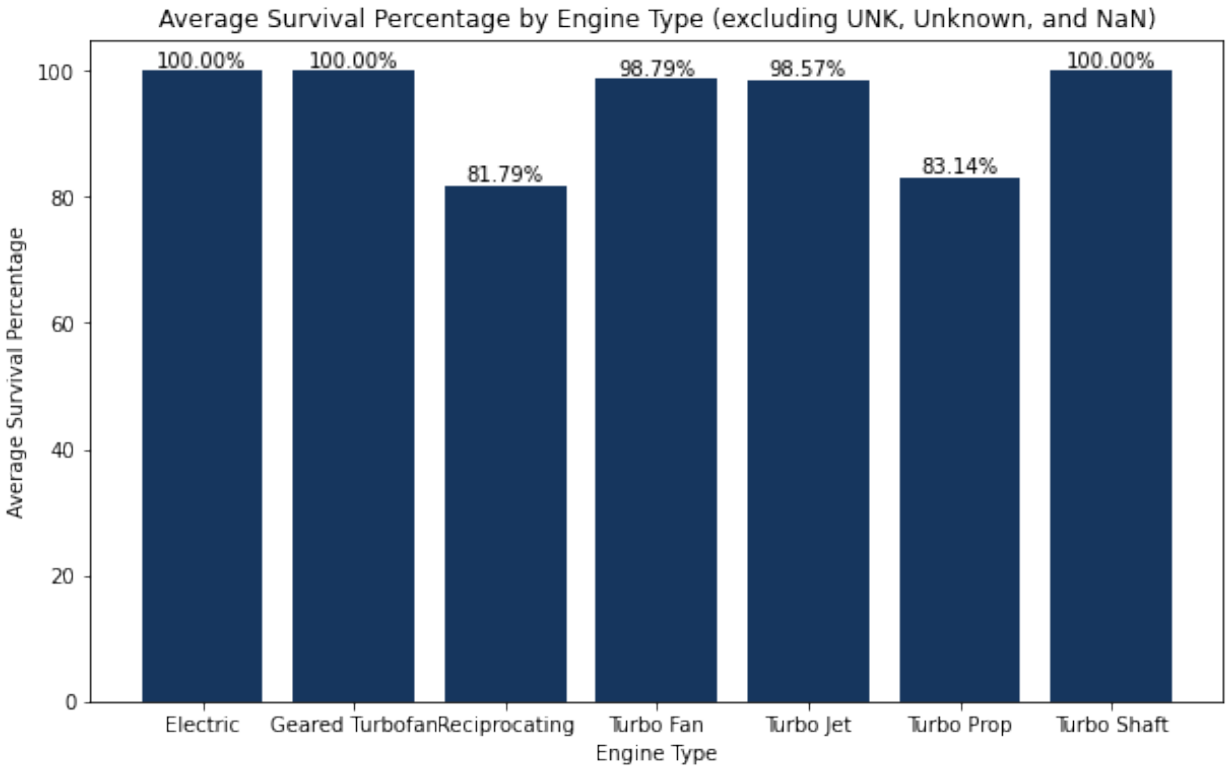
```
       Engine.Type  Number.of.Engines  Total.Fatal.Injuries  \
0         Electric                1.0                   0.0
1  Geared Turbofan                2.0                   0.0
2    Reciprocating            18815.0                5799.0
3        Turbo Fan             1450.0                 732.0
4        Turbo Jet              244.0                  76.0
5       Turbo Prop             1559.0                 755.0
6      Turbo Shaft                7.0                   0.0

   Total.Serious.Injuries  Total.Minor.Injuries  Total.Uninjured
0                     0.0                   0.0              2.0
1                     0.0                   0.0            121.0
2                  4211.0                3065.0          18766.0
3                   977.0                 425.0          58128.0
4                    80.0                  40.0           5109.0
5                   260.0                 189.0           3273.0
6                     0.0                   2.0             18.0
```

Out[43]:

	Engine.Type	Number.of.Engines	Total.Fatal.Injuries	Total.Serious.Injuries	Total.Minor.Injuries
0	Electric	1.0	0.0	0.0	0.0
1	Geared Turbofan	2.0	0.0	0.0	0.0
2	Reciprocating	18815.0	5799.0	4211.0	3065.0
3	Turbo Fan	1450.0	732.0	977.0	425.0

4	Turbo Jet	244.0	76.0	80.0	40.0
5	Turbo Prop	1559.0	755.0	260.0	189.0
6	Turbo Shaft	7.0	0.0	0.0	2.0

In [44]:
```
#Exclude specific values from the 'Engine.Type' column
excluded_values = ['UNK', 'Unknown', np.nan]
filtered_data = aviation_data_engine[~aviation_data_engine['Engine.Typ

#Group by 'Engine.Type' and calculate the average survival percentage
avg_survival_by_engine = filtered_data.groupby('Engine.Type')['Surviva

#Create a bar chart for the average survival percentage for each engin
plt.figure(figsize=(10, 6))
bar_plot = plt.bar(avg_survival_by_engine.index, avg_survival_by_engin

#Set the x-axis labels to be the engine types
plt.xticks(avg_survival_by_engine.index)
plt.xlabel('Engine Type')
plt.ylabel('Average Survival Percentage')
plt.title('Average Survival Percentage by Engine Type (excluding UNK,

#Annotate each bar with its corresponding value (survival percentage)
for bar in bar_plot:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height, f'{height:.2f}

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



This is really interesting data - especially when looked at with the dataframe. At first, it's easy to see that there are three types of engines with 100% survival rates however, the total number of passengers flown by all three of these engines combined doesn't come close to the number of passengers flown by engines with very slightly lower survival rates. Specifically,the Turbo Fan Engine flew 59,530 people. I would recommend investigating this further to determine that the fatalities were due to injuries from the accident and not for some other reason.

Recommendation 3:

In this design exploration, let's look at Aircraft.damage, anything destroyed is probably a high-risk, and minor being low-risk, substantial being medium risk so I will determine the average survival percentage by aircraft damage.

When considering an investment, do certain make/models tend to be destroyed when they are in accidents versus other aircrafts?

In [45]:
```
#View unique values listed in 'Aircraft.damage'
unique_aircraft_damage = aviation_data_clean['Aircraft.damage'].unique
unique_aircraft_damage
```

Out[45]:
```
array(['Substantial', 'Destroyed', 'Minor', nan, 'Unknown'], dtype=ob
ject)
```

I will again have to calculate the passenger survival percentage on the clean, disaggregated data.

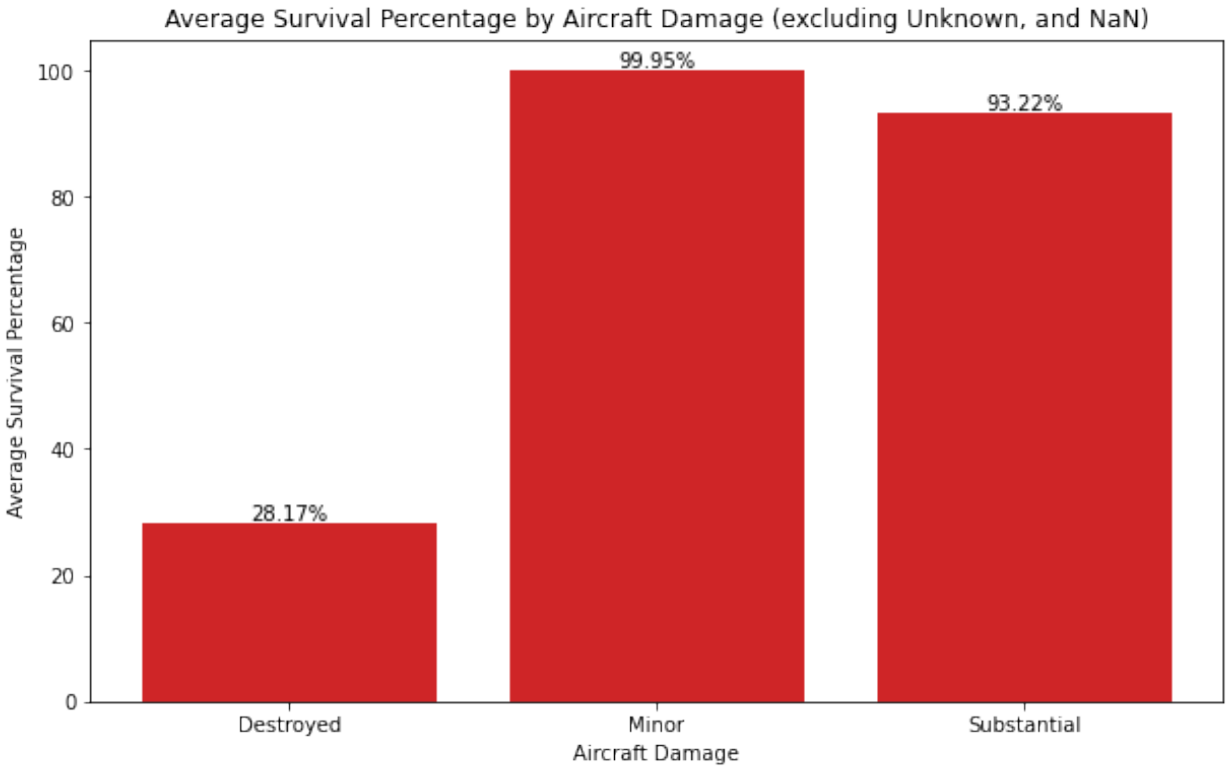
In [46]:
```
#Select only numeric columns for summing
passenger_injury_columns = aviation_data_clean.select_dtypes(include=[

#Drop rows with zero number of passengers; eliminate future NaN values
aviation_data_damage = aviation_data_clean[aviation_data_clean['Total.

#Group by 'Aircraft.damage' and sum the values for each group
aviation_data_damage = aviation_data_damage.groupby(['Aircraft.damage'

#Display the combined DataFrame
print(aviation_data_damage)

#Create a total passenger column
# Sum the 'Fatal', 'Serious', 'Minor', and 'Uninjured' columns for eac
aviation_data_damage['Total.Passengers'] = aviation_data_damage[['Tota
aviation_data_damage

#Create a survival percentage column
aviation_data_damage.loc[:, 'Total.Survivors'] = aviation_data_damage[
aviation_data_damage.loc[:, 'Survival.Percentage'] = (aviation_data_da

aviation_data_damage
```

```
  Aircraft.damage  Number.of.Engines  Total.Fatal.Injuries  \
0       Destroyed            3157.0                9452.0
1           Minor             876.0                  18.0
2     Substantial           19510.0                4080.0
3         Unknown              57.0                  46.0

   Total.Serious.Injuries  Total.Minor.Injuries  Total.Uninjured
0                   914.0                 938.0           1854.0
1                   148.0                 108.0          33169.0
2                  5274.0                3337.0          47460.0
3                    15.0                  20.0            463.0
```

Out[46]:

	Aircraft.damage	Number.of.Engines	Total.Fatal.Injuries	Total.Serious.Injuries	Total.Minor.Injur
0	Destroyed	3157.0	9452.0	914.0	93
1	Minor	876.0	18.0	148.0	10
2	Substantial	19510.0	4080.0	5274.0	333
3	Unknown	57.0	46.0	15.0	2

In [47]:
```
#Exclude specific values from the 'Aircraft.damage' column
excluded_values = ['UNK', 'Unknown', np.nan]
filtered_data = aviation_data_damage[~aviation_data_damage['Aircraft.d

#Group by 'Aircraft.damage' and calculate the average survival percent
avg_survival_by_damage = filtered_data.groupby('Aircraft.damage')['Sur

#Create a bar chart for the average survival percentage for each damag
plt.figure(figsize=(10, 6))
bar_plot = plt.bar(avg_survival_by_damage.index, avg_survival_by_damag

#Set the x-axis labels to be the damage category
plt.xticks(avg_survival_by_damage.index)
plt.xlabel('Aircraft Damage')
plt.ylabel('Average Survival Percentage')
plt.title('Average Survival Percentage by Aircraft Damage (excluding U

#Annotate each bar with its corresponding value (survival percentage)
for bar in bar_plot:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height, f'{height:.2f}

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



This bar chart shows that plane make/models where the damage value is 'destroyed' would be a high-risk investment because the have a low survival rate.

Operational Recommendation:

Let's take a look at another potential area of interest to investors. This data explores passenger survival percentage along with day of the week. This would be of interest to the operation costs of the airline. I will determine the average survival percentage by day of the week.

When considering this investment, which is the safest day of the week to fly?

In [48]:
```
aviation_data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 27586 entries, 20170710X52551 to 20221227106497
Data columns (total 30 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Investigation.Type      27586 non-null  object 
 1   Accident.Number         27586 non-null  object 
 2   Event.Date              27586 non-null  object 
 3   Location                27579 non-null  object 
 4   Country                 27579 non-null  object 
 5   Latitude                22082 non-null  object 
 6   Longitude               22073 non-null  object 
 7   Airport.Code            17756 non-null  object 
 8   Airport.Name            18236 non-null  object 
 9   Injury.Severity         26774 non-null  object 
 10  Aircraft.damage         26306 non-null  object 
 11  Aircraft.Category       27586 non-null  object 
 12  Registration.Number     27362 non-null  object 
 13  Make                    27586 non-null  object 
 14  Model                   27586 non-null  object 
 15  Amateur.Built           27569 non-null  object 
 16  Number.of.Engines       24836 non-null  float64
 17  Engine.Type             23373 non-null  object 
 18  FAR.Description         27087 non-null  object 
 19  Schedule                2983 non-null   object 
 20  Purpose.of.flight       23856 non-null  object 
 21  Air.carrier             11257 non-null  object 
 22  Total.Fatal.Injuries    24426 non-null  float64
 23  Total.Serious.Injuries  24369 non-null  float64
 24  Total.Minor.Injuries    24714 non-null  float64
 25  Total.Uninjured         26691 non-null  float64
 26  Weather.Condition       24542 non-null  object 
 27  Broad.phase.of.flight   6390 non-null   object 
 28  Report.Status           22624 non-null  object 
 29  Publication.Date        26587 non-null  object 
dtypes: float64(5), object(25)
memory usage: 6.5+ MB
```

In [49]:
```
aviation_data_combined.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7105 entries, 0 to 7269
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Make                   7105 non-null   object 
 1   Model                  7105 non-null   object 
 2   Total.Fatal.Injuries   7105 non-null   float64
 3   Total.Serious.Injuries 7105 non-null   float64
 4   Total.Minor.Injuries   7105 non-null   float64
 5   Total.Uninjured        7105 non-null   float64
 6   Total.Passengers       7105 non-null   float64
 7   Total.Survivors        7105 non-null   float64
 8   Survival.Percentage    7105 non-null   float64
dtypes: float64(7), object(2)
memory usage: 555.1+ KB
```

The data set provides information on the date of the accients so this can be used to determine which days of the week accidents are happening.

In [50]:
```
#Convert the 'Event_Date' column to datetime format
aviation_data_clean['Event.Date'] = pd.to_datetime(aviation_data_clean

#Create a new column 'day_of_week' to store the day of the week
aviation_data_clean['day_of_week'] = aviation_data_clean['Event.Date']

#Print the DataFrame to see the new 'day_of_week' column
print(aviation_data_clean.head())
```

```
                Investigation.Type Accident.Number Event.Date  \
Event.Id                                                        
20170710X52551          Accident      NYC79AA106 1979-09-17   
20020909X01562          Accident      SEA82DA022 1982-01-01   
20020909X01561          Accident      NYC82DA015 1982-01-01   
20020917X02148          Accident      FTW82FRJ07 1982-01-02   
20020917X02134          Accident      FTW82FRA14 1982-01-02   

                         Location        Country Latitude Longitude  \
Event.Id                                                             
20170710X52551        BOSTON, MA  United States  42.4453  -70.7583   
20020909X01562       PULLMAN, WA  United States      NaN       NaN   
20020909X01561  EAST HANOVER, NJ  United States      NaN       NaN   
20020917X02148         HOMER, LA  United States      NaN       NaN   
20020917X02134        HEARNE, TX  United States      NaN       NaN   

                Airport.Code        Airport.Name Injury.Severity  \
```

```
Event.Id
20170710X52551          NaN                  NaN       Non-Fatal
20020909X01562          NaN  BLACKBURN AG STRIP       Non-Fatal
20020909X01561          N58             HANOVER       Non-Fatal
20020917X02148          NaN                  NaN       Non-Fatal
20020917X02134          T72    HEARNE MUNICIPAL        Fatal(1)

               Aircraft.damage Aircraft.Category Registration.Number
\
Event.Id
20170710X52551     Substantial          Airplane              CF-TLU
20020909X01562     Substantial          Airplane              N2482N
20020909X01561     Substantial          Airplane              N7967Q
20020917X02148       Destroyed          Airplane              N14779
20020917X02134       Destroyed          Airplane              N758SK

                           Make   Model Amateur.Built  Number.of.En
gines  \
Event.Id
20170710X52551  MCDONNELL DOUGLAS     DC9            No
2.0
20020909X01562            CESSNA     140            No
1.0
20020909X01561            CESSNA    401B            No
2.0
20020917X02148          BELLANCA  17-30A            No
1.0
20020917X02134            CESSNA   R172K            No
1.0

                  Engine.Type            FAR.Description Schedule  \
Event.Id
20170710X52551      Turbo Fan          Part 129: Foreign     SCHD
20020909X01562  Reciprocating  Part 91: General Aviation      NaN
20020909X01561  Reciprocating  Part 91: General Aviation      NaN
20020917X02148  Reciprocating  Part 91: General Aviation      NaN
20020917X02134  Reciprocating  Part 91: General Aviation      NaN

               Purpose.of.flight Air.carrier  Total.Fatal.Injuries  \
Event.Id
20170710X52551              NaN  Air Canada                   NaN
20020909X01562         Personal         NaN                   0.0
20020909X01561         Business         NaN                   0.0
20020917X02148         Personal         NaN                   0.0
20020917X02134         Personal         NaN                   1.0

                Total.Serious.Injuries  Total.Minor.Injuries  Total.U
ninjured  \
Event.Id
20170710X52551                    NaN                   1.0
```

```
44.0
20020909X01562                      0.0                    0.0
2.0
20020909X01561                      0.0                    0.0
2.0
20020917X02148                      0.0                    1.0
0.0
20020917X02134                      0.0                    0.0
0.0

                Weather.Condition Broad.phase.of.flight   Report.Statu
s  \
Event.Id
20170710X52551               VMC                 Climb  Probable Caus
e
20020909X01562               VMC              Takeoff  Probable Caus
e
20020909X01561               IMC              Landing  Probable Caus
e
20020917X02148               IMC               Cruise  Probable Caus
e
20020917X02134               IMC              Takeoff  Probable Caus
e

                Publication.Date day_of_week
Event.Id
20170710X52551       19-09-2017      Monday
20020909X01562       01-01-1982      Friday
20020909X01561       01-01-1982      Friday
20020917X02148       02-01-1983    Saturday
20020917X02134       02-01-1983    Saturday
```

I will again have to calculate the passenger survival percentage on the clean, disaggregated data.

In [51]:
```
#Select only numeric columns for summing
passenger_injury_columns = aviation_data_clean.select_dtypes(include=[

#Drop rows with zero number of passengers; eliminate future NaN values
aviation_data_days = aviation_data_clean[aviation_data_clean['Total.Fa

#Group by 'day_of_week' and sum the values for each group
aviation_data_days = aviation_data_days.groupby(['day_of_week'], as_in

#Display the combined DataFrame
print(aviation_data_days)
```

```
  day_of_week  Number.of.Engines  Total.Fatal.Injuries  \
0      Friday             3670.0                1835.0
1      Monday             3158.0                1924.0
2    Saturday             4453.0                2756.0
3      Sunday             3906.0                2437.0
4    Thursday             3324.0                1620.0
5     Tuesday             2938.0                1268.0
6   Wednesday             3137.0                1848.0

   Total.Serious.Injuries  Total.Minor.Injuries  Total.Uninjured
0                  1011.0                 651.0          22233.0
1                   861.0                 532.0          22827.0
2                  1404.0                 894.0          21427.0
3                  1139.0                 810.0          19299.0
4                   703.0                 592.0          20479.0
5                   874.0                 653.0          19041.0
6                   880.0                 612.0          20155.0
```

In [52]:
```
#Drop Number.of.Engines Column
aviation_data_days.drop(columns = ['Number.of.Engines'], inplace=True
aviation_data_days.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7 entries, 0 to 6
Data columns (total 5 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   day_of_week            7 non-null      object
 1   Total.Fatal.Injuries   7 non-null      float64
 2   Total.Serious.Injuries  7 non-null      float64
 3   Total.Minor.Injuries   7 non-null      float64
 4   Total.Uninjured        7 non-null      float64
dtypes: float64(4), object(1)
memory usage: 336.0+ bytes
```

In [53]:
```
#Create a total passenger column
#Sum 'Fatal', 'Serious', 'Minor', and 'Uninjured' columns for each row
aviation_data_days['Total.Passengers'] = aviation_data_days[['Total.Fa
aviation_data_days
```

Out[53]:

	day_of_week	Total.Fatal.Injuries	Total.Serious.Injuries	Total.Minor.Injuries	Total.Uninjured	To
0	Friday	1835.0	1011.0	651.0	22233.0	
1	Monday	1924.0	861.0	532.0	22827.0	
2	Saturday	2756.0	1404.0	894.0	21427.0	
3	Sunday	2437.0	1139.0	810.0	19299.0	
4	Thursday	1620.0	703.0	592.0	20479.0	
5	Tuesday	1268.0	874.0	653.0	19041.0	
6	Wednesday	1848.0	880.0	612.0	20155.0	

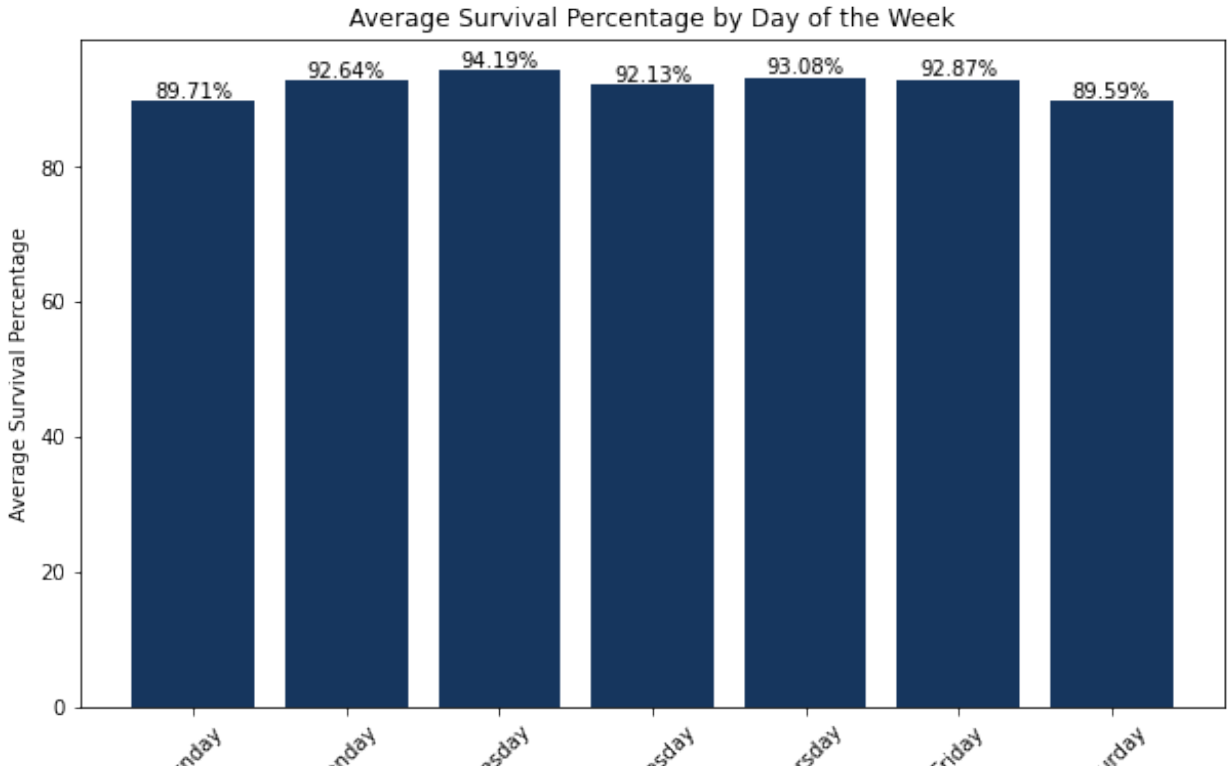
In [54]:
```
#Create a survival percentage column
aviation_data_days.loc[:, 'Total.Survivors'] = aviation_data_days['Tot
aviation_data_days.loc[:, 'Survival.Percentage'] = (aviation_data_days

aviation_data_days
```

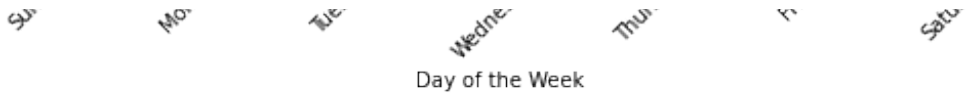
Out[54]:

	day_of_week	Total.Fatal.Injuries	Total.Serious.Injuries	Total.Minor.Injuries	Total.Uninjured	To
0	Friday	1835.0	1011.0	651.0	22233.0	
1	Monday	1924.0	861.0	532.0	22827.0	
2	Saturday	2756.0	1404.0	894.0	21427.0	
3	Sunday	2437.0	1139.0	810.0	19299.0	
4	Thursday	1620.0	703.0	592.0	20479.0	
5	Tuesday	1268.0	874.0	653.0	19041.0	
6	Wednesday	1848.0	880.0	612.0	20155.0	

In [55]:

```
import matplotlib.pyplot as plt

#Create a list of the days of the week in the desired order starting w
days_of_week_order = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thu

#Group by 'day_of_week' and calculate the average survival percentage
avg_survival_by_day = aviation_data_days.groupby('day_of_week')['Survi

#Reorder the days of the week in the desired order
avg_survival_by_day = avg_survival_by_day.reindex(days_of_week_order)

#Create a bar chart for the average survival percentage for each day o
plt.figure(figsize=(10, 6))
bar_plot = plt.bar(avg_survival_by_day.index, avg_survival_by_day.valu

#Set the x-axis labels to be the days of the week
plt.xticks(rotation=45)
plt.xlabel('Day of the Week')
plt.ylabel('Average Survival Percentage')
plt.title('Average Survival Percentage by Day of the Week')

#Annotate each bar with its corresponding value (survival percentage)
for bar in bar_plot:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height, f'{height:.2f}

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





This bar graph shows that the survival rate is higher, in some cases 5% higher, during weekday flights.

Conclusions

This analysis discovered that the safest airplane (lowest-risk) is the Boeing 777. This is based on the calculated passenger survival rate.

This analysis leads to three airplane design recommendations for a low-risk investment.

1. While experimental/amateur built aircrafts may show success, the survival rate for amateur built aircrafts involved in accidents is more than 15% lower than professional manufacturing. **Purchase professionally manufactured airplanes.**
2. Several types of engines showed reliability. While electric, geared turbofan, and turbo shaft all had 100% survival rate, their survivors combined were 143 people. The turbo fan engine has a 98.78% survival rate and transported over 60K passengers suggesting further investigation is needed into the reported deaths to confirm that they were a direct result of the airplane accident. **Consider further investigation of the turbo fan engine.**
3. Looking at the aircraft damage category, anything destroyed is a high-risk, substantial is medium- risk, and minor is low-risk. **Airplanes that sustain minor damage in accidents are low-risk**

This last recommendation is an operational recommendation around the best day of the week to schedule flights. The data showed the best survival rate on Tuesday. Overall, the data shows that it's safest to fly on weekdays. **Consider offering lower fares and promotions on weekdays when it is safer to fly.**