Final Project Submission

Please fill out:

- Student name: Tanveer Mbitiru Chege
- Student pace: part time
- Scheduled project review date/time:
- Instructor name: Fidelis Wanalenge

Step 1: Understanding the Data

Objective: Our company wants to enter the aviation business but needs to understand which aircraft types are lowest risk for their new venture.

Key Questions to Answer as a Company:

1. Which aircraft models have the lowest accident rates?
2. What factors contribute most to severe accidents?
3. Are there specific conditions (weather, flight phase) that lead to more accidents?

Step 2: Exploring Our Data

What we will do here is:

1. Import the relevant libraries.
2. Load the data into a pandas DataFrame.
3. Check basic info.
4. Identify missing values.

- Pandas will handle tabular data
- Numpy will handle the numerical data
- Matplotlib and seaborn will be for the visuals

Import the relevant Libraries

In [28]:
```
#importing the relevant libaries:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading the data into a pandas DataFrame

In [29]:
```
#Load the data into a pandas DataFrame

aviation_df = pd.read_csv("data/Aviation_Data.csv")
```

```
C:\Users\mbiti\anaconda3\envs\learn-env\lib\site-packages\IPython\core\inte
ractiveshell.py:3145: DtypeWarning: Columns (6,7,28) have mixed types.Speci
fy dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

So we see our data has incconsistent data types so no we force pandas to scan the full file before telling what data types are there.

In [30]:
```
#let us fix that by Setting low_memory=False forcing pandas to read the entir
aviation_df = pd.read_csv("data/Aviation_Data.csv", low_memory=False, index_c
#let us now display our data:
aviation_df
```

Out[30]:

Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	La
20001218X45444	Accident	SEA87LA080	1948-10-24	MOOSE CREEK, ID	United States	
20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPORT, CA	United States	
20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, VA	United States	36.9
20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, CA	United States	
20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, OH	United States	
...	...	...	...	...	...	
20221227106491	Accident	ERA23LA093	2022-12-26	Annapolis, MD	United States	
20221227106494	Accident	ERA23LA095	2022-12-26	Hampton, NH	United States	
20221227106497	Accident	WPR23LA075	2022-12-26	Payson, AZ	United States	34
20221227106498	Accident	WPR23LA076	2022-12-26	Morgan, UT	United States	
20221230106513	Accident	ERA23LA097	2022-12-29	Athens, GA	United States	

90348 rows × 30 columns

Check basic info

Here we want to know the size of the data set.

In [31]:
```
#Let us see how many rows and columns we have:
aviation_df.shape
```

Out[31]:
```
(90348, 30)
```

Now let us get a quick summary of the data frame

In [32]:
```
#her we look at the column names, their data types and the number of values i
aviation_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 90348 entries, 20001218X45444 to 20221230106513
Data columns (total 30 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Investigation.Type      90348 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        73659 non-null  object 
dtypes: float64(5), object(25)
memory usage: 21.4+ MB
```

In [33]:
```
#let us look at the first 5 rows:
aviation_df.head()
```

Out[33]:

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

5 rows × 30 columns

In [34]:
```
#let us look at the last 5 rows:
aviation_df.tail()
```

Out[34]:

Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latitude
20221227106491	Accident	ERA23LA093	2022-12-26	Annapolis, MD	United States	NaN
20221227106494	Accident	ERA23LA095	2022-12-26	Hampton, NH	United States	NaN
20221227106497	Accident	WPR23LA075	2022-12-26	Payson, AZ	United States	341525N
20221227106498	Accident	WPR23LA076	2022-12-26	Morgan, UT	United States	NaN
20221230106513	Accident	ERA23LA097	2022-12-29	Athens, GA	United States	NaN

5 rows × 30 columns

Now we need a summary statistics of our data that is mean, min, max, std etc. Here we can in a way detect anomalies, see the distribution of values, see our spread etc and we may get some insight.

In [35]:
```
#Let us get that summary
aviation_df.describe()
```

Out[35]:

	Number.of.Engines	Total.Fatal.Injuries	Total.Serious.Injuries	Total.Minor.Injuries	Total.Un
count	82805.000000	77488.000000	76379.000000	76956.000000	82977.
mean	1.146585	0.647855	0.279881	0.357061	5.
std	0.446510	5.485960	1.544084	2.235625	27.
min	0.000000	0.000000	0.000000	0.000000	0.
25%	1.000000	0.000000	0.000000	0.000000	0.
50%	1.000000	0.000000	0.000000	0.000000	1.
75%	1.000000	0.000000	0.000000	0.000000	2.
max	8.000000	349.000000	161.000000	380.000000	699.

Identify missing values

We count how many missing (NaN) values we have in each column. This way we know what column we will clean or impute.

In [36]:
```
#let us check for the null values in each column:
aviation_df.isna().sum()
```

Out[36]:
```
Investigation.Type           0
Accident.Number           1459
Event.Date                1459
Location                  1511
Country                   1685
Latitude                 55966
Longitude                55975
Airport.Code             40099
Airport.Name             37558
Injury.Severity           2459
Aircraft.damage           4653
Aircraft.Category        58061
Registration.Number       2776
Make                      1522
Model                     1551
Amateur.Built             1561
Number.of.Engines         7543
Engine.Type               8536
FAR.Description          58325
Schedule                 77766
Purpose.of.flight         7651
Air.carrier              73700
Total.Fatal.Injuries     12860
Total.Serious.Injuries   13969
Total.Minor.Injuries     13392
Total.Uninjured           7371
Weather.Condition         5951
Broad.phase.of.flight    28624
Report.Status             7840
Publication.Date         16689
dtype: int64
```

From above we cannot tell for sure which ones can we drop safely so now we can display the percentage of missing values in each column and sort it in descending order why? Arranged data we help prioritize columns to drop and clean.

In [37]:
```
# From the output above we see so many null values but we need to see how dee
#if we return the percentage of the missing values we can see the magnitude o
percent_missing = aviation_df.isnull().sum() * 100 / len(aviation_df)
percent_missing = percent_missing.sort_values(ascending = False)
percent_missing
```

Out[37]:
```
Schedule                 86.073848
Air.carrier              81.573471
FAR.Description          64.555939
Aircraft.Category        64.263736
Longitude                61.954886
Latitude                 61.944924
Airport.Code             44.382831
Airport.Name             41.570372
Broad.phase.of.flight    31.681941
Publication.Date         18.471909
Total.Serious.Injuries   15.461327
Total.Minor.Injuries     14.822686
Total.Fatal.Injuries     14.233851
Engine.Type               9.447913
Report.Status             8.677558
Purpose.of.flight         8.468367
Number.of.Engines         8.348829
Total.Uninjured           8.158454
Weather.Condition         6.586753
Aircraft.damage           5.150086
Registration.Number       3.072564
Injury.Severity           2.721698
Country                   1.865011
Amateur.Built             1.727764
Model                     1.716695
Make                      1.684597
Location                  1.672422
Event.Date                1.614867
Accident.Number           1.614867
Investigation.Type        0.000000
dtype: float64
```

Step 3: Dropping Our Data

We can safely drop columns with more than 50% missing values. Now to reduce some hard coding we can store the cleaned version in a new data frame and in our case aviation_df_clean.

In [38]:
```
# Drop columns with more than 50% missing values
threshold = 0.5
aviation_df_clean = aviation_df.loc[:, aviation_df.isnull().mean() < threshol
```

So we want to uniquely identify an accident so we can use key identifier and in this case Event.Date, Location and Make.

In [39]:
```
# Drop duplicate rows based on key identifiers
aviation_df_clean.drop_duplicates(subset=['Event.Date', 'Location', 'Make'],
```

```
<ipython-input-39-2952a22f3abc>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  aviation_df_clean.drop_duplicates(subset=['Event.Date', 'Location', 'Make'], inplace=True)
```

Now having done that we know we may have disordered index values. So now to have a conistent flowing layout I will reset my index

In [40]:
```
# Reset index after dropping
aviation_df_clean.reset_index(drop=True, inplace=True)
```

Now we need to have a datetime that is in the correct format so that we can work with it so here we will use Event.Date to do so. The to ensure that in our Event.Date column the invalid dates are put in a valid format I will write error = 'coerce' and it will convert invalid dates to NaT. So now we can do some sorting and filtering with this.

In [41]:
```
# Convert Event.Date column to datetime format
aviation_df_clean['Event.Date'] = pd.to_datetime(aviation_df_clean['Event.Dat
```

```
<ipython-input-41-2b329c00cbb6>:2: 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/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  aviation_df_clean['Event.Date'] = pd.to_datetime(aviation_df_clean['Event.Date'], errors='coerce')
```

In [42]:
```
#so now let us see our cleaned data frame has how many rows and columns
aviation_df_clean.shape
```

Out[42]: (88562, 24)

In [43]:
```
#Let us look at the columns first
print(aviation_df.columns)
```

```
Index(['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')
```

Handling the missing values

In [44]:
```
#Let us see the percentage of the missing values in our columns:
percent_missing
```

Out[44]:
```
Schedule                86.073848
Air.carrier             81.573471
FAR.Description         64.555939
Aircraft.Category       64.263736
Longitude               61.954886
Latitude                61.944924
Airport.Code            44.382831
Airport.Name            41.570372
Broad.phase.of.flight   31.681941
Publication.Date        18.471909
Total.Serious.Injuries  15.461327
Total.Minor.Injuries    14.822686
Total.Fatal.Injuries    14.233851
Engine.Type              9.447913
Report.Status            8.677558
Purpose.of.flight        8.468367
Number.of.Engines        8.348829
Total.Uninjured          8.158454
Weather.Condition        6.586753
Aircraft.damage          5.150086
Registration.Number      3.072564
Injury.Severity          2.721698
Country                  1.865011
Amateur.Built            1.727764
Model                    1.716695
Make                     1.684597
Location                 1.672422
Event.Date               1.614867
Accident.Number          1.614867
Investigation.Type       0.000000
dtype: float64
```

Dropping Columns

Here I realized I did not drop alot of things so now I had to do it one by one. Criteria Used: High missing values (>80% and >40%): Columns like Air.carrier, Schedule, Aircraft.Category, etc., were dropped because they had too many nulls (making them unreliable for analysis) Irrelevance to analysis: Columns like Longitude, Latitude, FAR.Description were thought to be unimportant for their specific goals.

In [45]:
```
# Drop columns with high missing values or irrelevant information
columns_to_drop = [
    'Air.carrier',          # >80% missing
    'Schedule',             # >80% missing
    'Aircraft.Category',    # Irrelevant + >40% missing
    'Airport.Code',         # Irrelevant + >40% missing
    'Airport.Name',         # Irrelevant + >40% missing
    'Longitude',            # >60% was missing
    'Latitude',             # >60% was missing
    'FAR.Description',      # >60% missing
    'Publication.Date',     # Irrelevant for my analysis
    'Registration.Number', # not useful for my analysis
    'Country',              # All "United States" (redundant)
    'Location',             # did see need for it
    'Report.Status',        # did not see the need
    'Weather.Condition'     # High missing values
]

aviation_df = aviation_df.drop(columns=columns_to_drop)
```

In [46]:
```
# Drop rows where 'Broad.phase.of.flight' as it is 30% missing)
aviation_df = aviation_df.dropna(subset=['Broad.phase.of.flight'])

# Check remaining missing values
print("Remaining missing values per column:")
print(aviation_df.isnull().sum())
```

```
Remaining missing values per column:
Investigation.Type            0
Accident.Number               0
Event.Date                    0
Injury.Severity               0
Aircraft.damage            1458
Make                         11
Model                        30
Amateur.Built                19
Number.of.Engines           922
Engine.Type                 400
Purpose.of.flight          1097
Total.Fatal.Injuries      10666
Total.Serious.Injuries    11389
Total.Minor.Injuries      10745
Total.Uninjured            5005
Broad.phase.of.flight         0
dtype: int64
```

Now I need to convert Tota fatal injuries to a number: 1. I need to fillna missing values with 0 why I am assuming no fatalities if they were not reported. 2. I will convert the values to integer for consistency purposes.

In [47]:
```
# Convert 'Total.Fatal.Injuries' to numeric values
aviation_df_clean['Total.Fatal.Injuries'] = pd.to_numeric(aviation_df_clean['

# Fill missing values with 0 (assumes missing = no fatalities)
aviation_df_clean['Total.Fatal.Injuries'].fillna(0, inplace=True)

# Convert to integer
aviation_df_clean['Total.Fatal.Injuries'] = aviation_df_clean['Total.Fatal.Ir
```

```
<ipython-input-47-64f42554f602>:2: 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/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  aviation_df_clean['Total.Fatal.Injuries'] = pd.to_numeric(aviation_df_cle
an['Total.Fatal.Injuries'], errors='coerce')
C:\Users\mbiti\anaconda3\envs\learn-env\lib\site-packages\pandas\core\serie
s.py:4517: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  return super().fillna(
<ipython-input-47-64f42554f602>:8: 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/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  aviation_df_clean['Total.Fatal.Injuries'] = aviation_df_clean['Total.Fata
l.Injuries'].astype(int)
```

To easily see injuries as they were categoriesed I can put then in one box, injury_cols.So now with that I can clean them together:

1. I will convert them to numeric type
2. I then replace the missing values with 0
3. I will convert the integer for uniformity purposes.

In [48]:
```
injury_cols = ['Total.Serious.Injuries', 'Total.Minor.Injuries', 'Total.Uninj

for col in injury_cols:
    aviation_df[col] = pd.to_numeric(aviation_df[col], errors='coerce').fillr
```

Now let us have a new column we will call it Total.Occupants where inside it I will sum up everyone involved in the accident that is Fatalities, Serious injuries, minor injuries and the Uninjured. So this tells me how many people were on board on each aircraft

In [49]:
```
aviation_df_clean['Total.Occupants'] = (
    aviation_df_clean['Total.Fatal.Injuries'] +
    aviation_df_clean['Total.Serious.Injuries'] +
    aviation_df_clean['Total.Minor.Injuries'] +
    aviation_df_clean['Total.Uninjured']
)
```

```
<ipython-input-49-855c2f6952f0>:1: 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/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  aviation_df_clean['Total.Occupants'] = (
```

Now I need to classify fatal and non-fatal events so I need a new column I will call it Was.Fatal and I will say 1 if we had atleast 1 fatality and 0 if no fatalities.

In [50]:
```
aviation_df['Was.Fatal'] = aviation_df['Total.Fatal.Injuries'].apply(lambda x
```

In [51]:
```
# Convert 'Event.Date' to datetime
aviation_df['Event.Date'] = pd.to_datetime(aviation_df['Event.Date'])

# Extract year/month/day for analysis
aviation_df['Year'] = aviation_df['Event.Date'].dt.year
aviation_df['Month'] = aviation_df['Event.Date'].dt.month
```

Now let us change the text formats for some key categorical columns to standadersized format eg we want takeoff to be Takeoff. While we are here we can fix some inconsistencies in our data.

In [52]:
```
# Standardize text casing in categorical columns
text_columns = ['Broad.phase.of.flight', 'Make', 'Model', 'Purpose.of.flight'
for col in text_columns:
    aviation_df[col] = aviation_df[col].str.title()

# Fix common inconsistencies
aviation_df['Make'] = aviation_df['Make'].replace({
    'Mcdonnell Douglas': 'McDonnell Douglas',
    'Bell Helicopter': 'Bell'
})
```

Now let us see how the number of engines are distribued in our data that way we can check for abnormal values.

In [53]:
```
# Check for outliers in 'Number.of.Engines'
print(aviation_df['Number.of.Engines'].value_counts())
```

```
1.0    51236
2.0     8056
0.0      743
3.0      442
4.0      325
Name: Number.of.Engines, dtype: int64
```

Now let us see some stats for the total fatal injuries so that we understand the spread.

In [54]:
```
# Check fatal injuries distribution
print(aviation_df['Total.Fatal.Injuries'].describe())
```

```
count    51058.000000
mean         0.480219
std          2.833237
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max        265.000000
Name: Total.Fatal.Injuries, dtype: float64
```

In [55]:
```
#Let us see the percentage of the missing values in our columns:
print(aviation_df.isnull().sum())
```

```
Investigation.Type            0
Accident.Number               0
Event.Date                    0
Injury.Severity               0
Aircraft.damage            1458
Make                         11
Model                        30
Amateur.Built                19
Number.of.Engines           922
Engine.Type                 400
Purpose.of.flight          1097
Total.Fatal.Injuries      10666
Total.Serious.Injuries        0
Total.Minor.Injuries          0
Total.Uninjured               0
Broad.phase.of.flight         0
Was.Fatal                     0
Year                          0
Month                         0
dtype: int64
```

From my output above I see we still have so many nulls so what do I do? I will fill missing values in the categorical columns with 'Unknown' then fillna Number.of.Engines with the median. Why because it looks to be less sensistive to outliers.

In [56]:
```
# Fill common categorical nulls with 'Unknown'
aviation_df['Aircraft.damage'].fillna('Unknown', inplace=True)
aviation_df['Make'].fillna('Unknown', inplace=True)
aviation_df['Model'].fillna('Unknown', inplace=True)
aviation_df['Amateur.Built'].fillna('Unknown', inplace=True)
aviation_df['Engine.Type'].fillna('Unknown', inplace=True)
aviation_df['Purpose.of.flight'].fillna('Unknown', inplace=True)

# For numerical fields like Number.of.Engines, use median (or mode)
aviation_df['Number.of.Engines'].fillna(aviation_df['Number.of.Engines'].medi

# If you haven't already done this for fatalities:
aviation_df['Total.Fatal.Injuries'] = pd.to_numeric(aviation_df['Total.Fatal.
```

In [57]:
```
#let us see the percentage of the missing values in our columns:
print(aviation_df.isnull().sum())
```

```
Investigation.Type         0
Accident.Number            0
Event.Date                 0
Injury.Severity            0
Aircraft.damage            0
Make                       0
Model                      0
Amateur.Built              0
Number.of.Engines          0
Engine.Type                0
Purpose.of.flight          0
Total.Fatal.Injuries       0
Total.Serious.Injuries     0
Total.Minor.Injuries       0
Total.Uninjured            0
Broad.phase.of.flight      0
Was.Fatal                  0
Year                       0
Month                      0
dtype: int64
```

In [58]:

```
# Verify cleaned data
print("\nFinal DataFrame Info:")
aviation_df.info()

# Sample output
print("\nSample Data:")
print(aviation_df.head(3))
```

```
Final DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
Index: 61724 entries, 20001218X45444 to 20080125X00106
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Investigation.Type      61724 non-null  object
 1   Accident.Number         61724 non-null  object
 2   Event.Date              61724 non-null  datetime64[ns]
 3   Injury.Severity         61724 non-null  object
 4   Aircraft.damage         61724 non-null  object
 5   Make                    61724 non-null  object
 6   Model                   61724 non-null  object
 7   Amateur.Built           61724 non-null  object
 8   Number.of.Engines       61724 non-null  float64
 9   Engine.Type             61724 non-null  object
 10  Purpose.of.flight       61724 non-null  object
 11  Total.Fatal.Injuries    61724 non-null  int32
 12  Total.Serious.Injuries  61724 non-null  int32
 13  Total.Minor.Injuries    61724 non-null  int32
 14  Total.Uninjured         61724 non-null  int32
 15  Broad.phase.of.flight   61724 non-null  object
 16  Was.Fatal               61724 non-null  int64
 17  Year                    61724 non-null  int64
 18  Month                   61724 non-null  int64
dtypes: datetime64[ns](1), float64(1), int32(4), int64(3), object(10)
memory usage: 8.5+ MB

Sample Data:
              Investigation.Type Accident.Number Event.Date Injury.Severit
y  \
Event.Id
20001218X45444           Accident      SEA87LA080 1948-10-24        Fatal
(2)
20001218X45447           Accident      LAX94LA336 1962-07-19        Fatal
(4)
20061025X01555           Accident      NYC07LA005 1974-08-30        Fatal
(3)

              Aircraft.damage     Make     Model Amateur.Built  \
Event.Id
20001218X45444       Destroyed  Stinson     108-3            No
20001218X45447       Destroyed    Piper  Pa24-180            No
20061025X01555       Destroyed   Cessna      172M            No

               Number.of.Engines    Engine.Type Purpose.of.flight  \
Event.Id
20001218X45444                1.0  Reciprocating          Personal
20001218X45447                1.0  Reciprocating          Personal
20061025X01555                1.0  Reciprocating          Personal

               Total.Fatal.Injuries  Total.Serious.Injuries  \
Event.Id
20001218X45444                     2                       0
20001218X45447                     4                       0
20061025X01555                     3                       0
```

```
                Total.Minor.Injuries  Total.Uninjured Broad.phase.of.flight
\
Event.Id
20001218X45444                     0                0               Cruise
20001218X45447                     0                0              Unknown
20061025X01555                     0                0               Cruise

                Was.Fatal  Year  Month
Event.Id
20001218X45444          1  1948     10
20001218X45447          1  1962      7
20061025X01555          1  1974      8
```

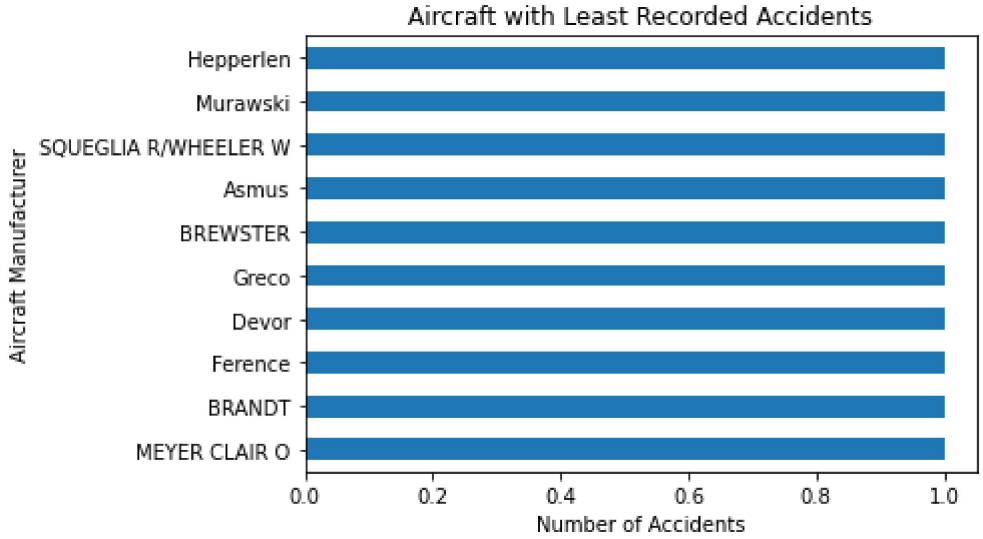
In [59]:
```
aviation_df['Total.Occupants'] = (
    aviation_df['Total.Fatal.Injuries'] +
    aviation_df['Total.Serious.Injuries'] +
    aviation_df['Total.Minor.Injuries'] +
    aviation_df['Total.Uninjured']
)
```

In [60]:
```
aviation_df['Fatality'] = aviation_df['Total.Fatal.Injuries']
```

Step 4: Exploratory Data Analysis (EDA)

Here now I analyze patterns: So let us start by seeing the 10 aircraft manafucturer with the least number of accidents why because that tells us which manufacturers are reliable.

In [61]:
```
top_safe_models = aviation_df_clean['Make'].value_counts().tail(10)
top_safe_models.plot(kind='barh', title='Aircraft with Least Recorded Acciden
plt.xlabel("Number of Accidents")
plt.ylabel("Aircraft Manufacturer")
plt.show()
```



Here now we want a percentage breakdown of fatal and non-fatal accidents.

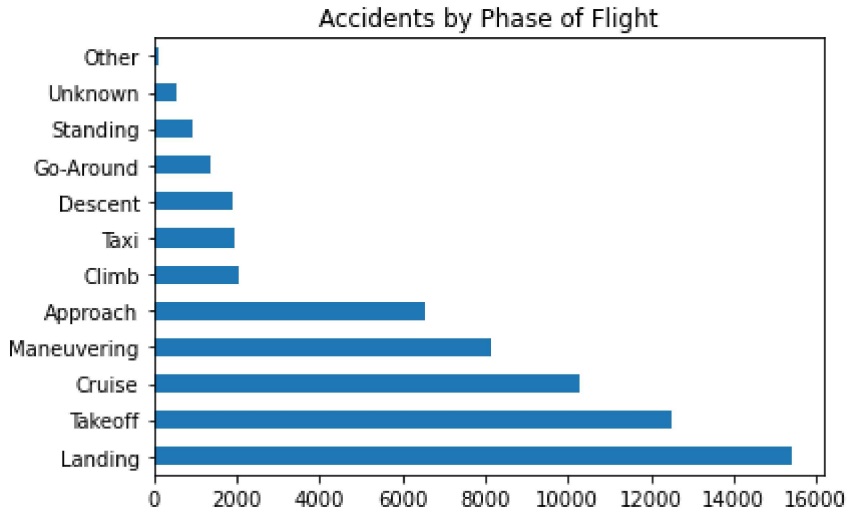
In [62]:
```
aviation_df['Was.Fatal'].value_counts(normalize=True) * 100
```

Out[62]:
```
0    81.145098
1    18.854902
Name: Was.Fatal, dtype: float64
```

Now that we have seen how fatal accidents were we need to see how these accidents are distrubuted across the different stages of a flight eg was it during takeoff or landing why? So that we know where to focus more as a company and where to be keen about.

In [63]:
```
aviation_df['Broad.phase.of.flight'].value_counts().plot(kind='barh', title='
```

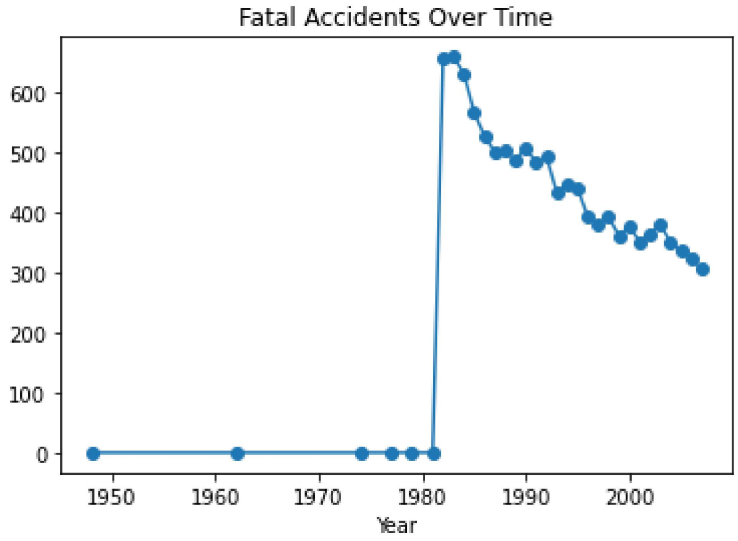
Out[63]:
```
<AxesSubplot:title={'center':'Accidents by Phase of Flight'}>
```



After that now that we need to see when were accidents fatal. If we see a spike in a certain year we can trace historical data and know why was there a spike.

In [64]:
```
aviation_df.groupby('Year')['Was.Fatal'].sum().plot(kind='line', marker='o',
```

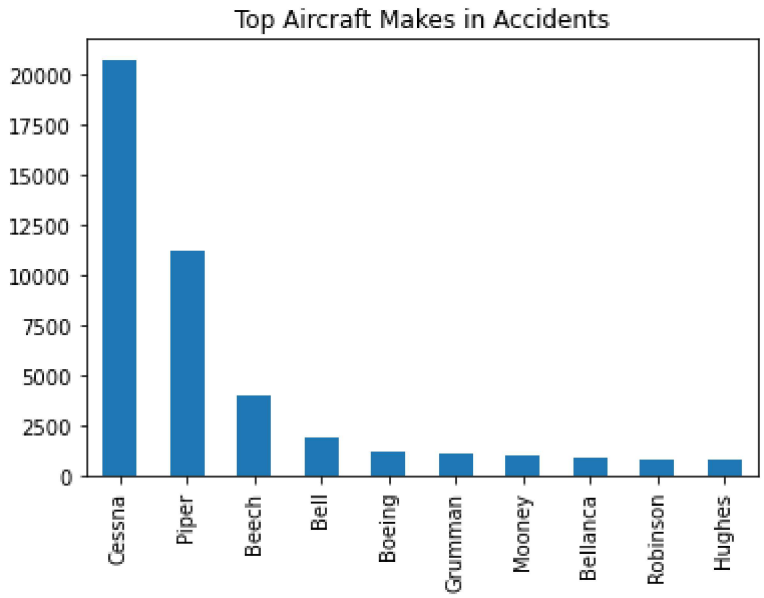
Out[64]: <AxesSubplot:title={'center':'Fatal Accidents Over Time'}, xlabel='Year'>



After that now what do we do? we ask ourselves now amongst these manufacturers should we avoid? Why we know more accidents have happened during cruise, takeoff and landing so with that we know there is an element of the aircraft in question, so we need to see which ones have been ivolved more so let us get the top 10.

In [65]:
```
aviation_df['Make'].value_counts().head(10).plot(kind='bar', title='Top Aircr
```

Out[65]: <AxesSubplot:title={'center':'Top Aircraft Makes in Accidents'}>



so we see the cessna is a no go craft. Next let us have the statistics of the key injury related columns.

In [66]:
```
aviation_df[['Total.Occupants', 'Fatality', 'Total.Serious.Injuries', 'Total.
```

Out[66]:

	Total.Occupants	Fatality	Total.Serious.Injuries	Total.Minor.Injuries
count	61724.000000	61724.000000	61724.000000	61724.000000
mean	5.221113	0.397236	0.198707	0.335769
std	24.942617	2.583226	0.780369	1.605021
min	0.000000	0.000000	0.000000	0.000000
25%	1.000000	0.000000	0.000000	0.000000
50%	2.000000	0.000000	0.000000	0.000000
75%	2.000000	0.000000	0.000000	0.000000
max	699.000000	265.000000	81.000000	171.000000

With this now we need a new column, a Fatality.Rate column why? Because we need to see the percentage of Fatalities per flight.

In [67]:
```
aviation_df['Fatality.Rate'] = (aviation_df['Fatality'] / aviation_df['Total.
aviation_df['Fatality.Rate'] = aviation_df['Fatality.Rate'].round(2)
aviation_df
```

Out[67]:

Event.Id	Investigation.Type	Accident.Number	Event.Date	Injury.Severity	Aircraft.damag
20001218X45444	Accident	SEA87LA080	1948-10-24	Fatal(2)	Destroye
20001218X45447	Accident	LAX94LA336	1962-07-19	Fatal(4)	Destroye
20061025X01555	Accident	NYC07LA005	1974-08-30	Fatal(3)	Destroye
20001218X45448	Accident	LAX96LA321	1977-06-19	Fatal(2)	Destroye
20041105X01764	Accident	CHI79FA064	1979-08-02	Fatal(1)	Destroye
...	...	...	...	...	
20071231X02014	Accident	DFW08FA053	2007-12-29	Fatal(1)	Destroye
20080109X00032	Accident	NYC08FA071	2007-12-30	Fatal(3)	Substanti
20080114X00045	Accident	LAX08FA043	2007-12-30	Fatal(1)	Substanti
20080129X00122	Accident	CHI08CA057	2007-12-30	Non-Fatal	Substanti
20080125X00106	Accident	SEA08CA056	2007-12-31	Non-Fatal	Substanti

61724 rows × 22 columns

Our data is too big we need to see a pattern so let us get the top 10 accidents with the highest

In [68]:
```
aviation_df.sort_values(by='Fatality', ascending=False).head(10)
```

Out[68]:

Event.Id	Investigation.Type	Accident.Number	Event.Date	Injury.Severity	Aircraft.damag
20011130X02321	Accident	DCA02MA001	2001-11-12	Fatal(265)	Destroye
20001208X06204	Accident	DCA96MA070	1996-07-17	Fatal(230)	Destroye
20001208X08606	Accident	DCA97MA058	1997-08-06	Fatal(228)	Destroye
20001213X31759	Accident	DCA87MA046	1987-08-16	Fatal(156)	Destroye
20020917X03104	Accident	DCA82AA028	1982-07-09	Fatal(153)	Destroye
20001214X37434	Accident	DCA85AA031	1985-08-02	Fatal(135)	Destroye
20001206X02233	Accident	DCA94MA076	1994-09-08	Fatal(132)	Destroye
20001213X28786	Accident	DCA89MA063	1989-07-19	Fatal(111)	Destroye
20001208X05743	Accident	DCA96MA054	1996-05-11	Fatal(110)	Destroye
20020123X00103	Accident	DCA01MA060	2001-09-11	Fatal(92)	Destroye

10 rows × 22 columns

So we know the most severe next is how do we see where most or all passengers died? Here is where our Fatality. Rate column comes in, it gets us our top 10 accidents with the highest fatality.

In [69]:
```
aviation_df.sort_values(by='Fatality.Rate', ascending=False).head(10)
```

Out[69]:

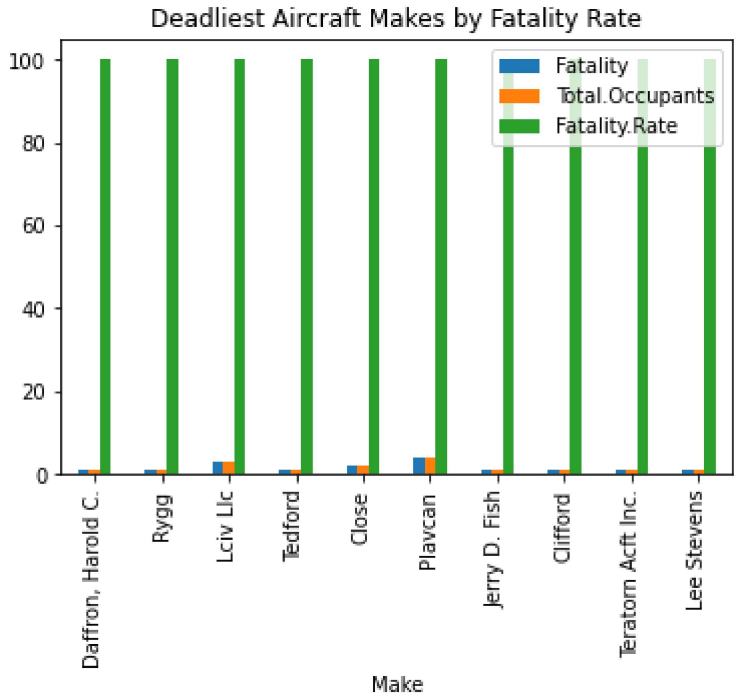
Event.Id	Investigation.Type	Accident.Number	Event.Date	Injury.Severity	Aircraft.damag
20001218X45444	Accident	SEA87LA080	1948-10-24	Fatal(2)	Destroye
20001213X35406	Accident	MIA87MA064	1986-12-31	Fatal(6)	Destroye
20001212X21756	Accident	LAX00FA319	2000-08-29	Fatal(2)	Destroye
20001213X35375	Accident	LAX87FA069	1986-12-27	Fatal(6)	Destroye
20001213X35338	Accident	DEN87FA035	1986-12-27	Fatal(1)	Destroye
20001212X21757	Accident	LAX00FA320	2000-08-29	Fatal(2)	Destroye
20001213X35298	Accident	ATL87FA051	1986-12-27	Fatal(2)	Substanti
20001213X35339	Accident	DEN87FA036	1986-12-29	Fatal(2)	Destroye
20001212X21755	Accident	LAX00FA314	2000-08-27	Fatal(1)	Destroye
20001213X35358	Accident	FTW87FA041	1986-12-30	Fatal(2)	Destroye

10 rows × 22 columns

Then everything we have done we put it in one basket, we take our total fatalities and total occupants in each Make. Then we see the total Fatality rate per make. This helps us find the top 10 most dangerous aircraft.

In [70]:
```
# Step 5: Grouped fatality rate by Make
make_risk = aviation_df.groupby('Make')[['Fatality', 'Total.Occupants']].sum(
make_risk['Fatality.Rate'] = (make_risk['Fatality'] / make_risk['Total.Occupa
make_risk = make_risk.sort_values(by='Fatality.Rate', ascending=False)
make_risk.head(10).plot(kind='bar', title='Deadliest Aircraft Makes by Fatali
```

Out[70]:
```
<AxesSubplot:title={'center':'Deadliest Aircraft Makes by Fatality Rate'},
xlabel='Make'>
```



So we get ready for tableau,let us see the list of column names in our cleaned dataset before we go visualise this on tableau

In [71]:
```
print(aviation_df.columns.tolist())
```

```
['Investigation.Type', 'Accident.Number', 'Event.Date', 'Injury.Severity',
'Aircraft.damage', 'Make', 'Model', 'Amateur.Built', 'Number.of.Engines',
'Engine.Type', 'Purpose.of.flight', 'Total.Fatal.Injuries', 'Total.Serious.
Injuries', 'Total.Minor.Injuries', 'Total.Uninjured', 'Broad.phase.of.fligh
t', 'Was.Fatal', 'Year', 'Month', 'Total.Occupants', 'Fatality', 'Fatality.
Rate']
```

Came back to clean formatting inconsistencies in the categorical data after I saw them in a bigger view. So I changed them to a title case.

In [66]:
```
# Standardizing categorical values for the existing columns
aviation_df['Injury.Severity'] = aviation_df['Injury.Severity'].str.strip().s
aviation_df['Aircraft.damage'] = aviation_df['Aircraft.damage'].str.strip().s
aviation_df['Broad.phase.of.flight'] = aviation_df['Broad.phase.of.flight'].s
aviation_df['Engine.Type'] = aviation_df['Engine.Type'].str.strip().str.title
aviation_df['Purpose.of.flight'] = aviation_df['Purpose.of.flight'].str.strip
```

So that I can be able to use my cleaned data on tableau I have to save it as a cleaned data set in my csv file.

In [72]:
```
aviation_df.to_csv("cleaned_aviation_data.csv", index=False)
```

In [ ]:
```

```