Final Project Submission

Please fill out:

- Student name: Abdirahman Abdi
- Student pace: Part-Time
- Instructor name: Williiam Okomba
- Blog post URL:https://github.com/Abdi-278/phase-1-project

CRISP_DM

- In this project i will be using the crisp-Dm methodology to find the insights and solution for the company
- I will be using these three steps to do it so far: 1.Business Understanding 2.Data Understanding 3.Data Preparation

Business Understanding

- The company aims to successfully expand into the aviation industry by purchasing and operating aircraft for commercial and private enterprises.
- Goal:To identify the aircraft models that present the lowest risk in terms of safety, reliability, and operational costs, thereby minimizing potential liabilities and maximizing profitability

Business Problem

Your company is expanding in to new industries to diversify its portfolio. Specifically, they are interested in purchasing and operating airplanes for commercial and private enterprises, but do not know anything about the potential risks of aircraft. You are charged with determining which aircraft are the lowest risk for the company to start this new business endeavor. You must then translate your findings into actionable insights that the head of the new aviation division can use to help decide which aircraft to purchase

In [424…

```
# The Data
#The data we were given were pulled from:
#kaggle datasets download -d khsamaha/aviation-accident-database-synopses
```

Data Preparation and Cleaning

Objectives

1. Load files using python packages

2. Look at information about data and column
3. Fix any missing or incorrect value
4. Ensure wanted observations are well structured.

Loading Python packages

In [425…
```
# numpy for high level mathematical functions and working with Arrays
import numpy as np
# pandas data manipulation and analysis for tablular data
import pandas as pd
# seaborn and matplotlib for data visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
 # Import the colormap module
import matplotlib.cm as cm
```

Loading the data

In [426…
```
#loading the data
df_original=pd.read_csv('AviationData.csv',encoding='mac_roman')
df_1_original=pd.read_csv('USState_Codes.csv')
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\1058131743.py:2: DtypeWarning: C
olumns (6,7,28) have mixed types. Specify dtype option on import or set low_memory
=False.
  df_original=pd.read_csv('AviationData.csv',encoding='mac_roman')
```

In [427…
```
# before doing anything else I create a copy of may df_original to keep the origina
df = df_original.copy()
df_1=df_1_original.copy()
```

Inspecting the Data

In [428…
```
#columns check
df.columns
```

Out[428]:
```
Index(['Event.Id', 'Investigation.Type', 'Accident.Number', 'Event.Date',
       'Location', 'Country', 'Latitude', 'Longitude', 'Airport.Code',
       'Airport.Name', 'Injury.Severity', 'Aircraft.damage',
       'Aircraft.Category', 'Registration.Number', 'Make', 'Model',
       'Amateur.Built', 'Number.of.Engines', 'Engine.Type', 'FAR.Description',
       'Schedule', 'Purpose.of.flight', 'Air.carrier', 'Total.Fatal.Injuries',
       'Total.Serious.Injuries', 'Total.Minor.Injuries', 'Total.Uninjured',
       'Weather.Condition', 'Broad.phase.of.flight', 'Report.Status',
       'Publication.Date'],
      dtype='object')
```

In [429…
```
#columns check
df_1.columns
```

Out[429]:
```
Index(['US_State', 'Abbreviation'], dtype='object')
```

In [430…
```
#if i want to display the first five rows of the dataset
df.head()
```

Out[430]:

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latitu
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MOOSE CREEK, ID	United States	N
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPORT, CA	United States	N
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, VA	United States	
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, CA	United States	N
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, OH	United States	N

5 rows × 31 columns

In [431…
```
df_1.head()
```

Out[431]:

	US_State	Abbreviation
0	Alabama	AL
1	Alaska	AK
2	Arizona	AZ
3	Arkansas	AR
4	California	CA

In [432…
```
# if i want to display the last five rows of the datasets
df.tail()
```

Out[432]:

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

5 rows × 31 columns

In [433…
```
df_1.tail()
```

Out[433]:

	US_State	Abbreviation
57	Virgin Islands	VI
58	Washington_DC	DC
59	Gulf of mexico	GM
60	Atlantic ocean	AO
61	Pacific ocean	PO

In [434…
```
#shape of the data
df.shape
```

Out[434]:
```
(88889, 31)
```

In [435…
```
#shape of the data
df_1.shape
```

Out[435]:
```
(62, 2)
```

In [436…
```
#Displaying how many rows and columns using the f string
print(f'the dataset has {df.shape[0]} rows and {df.shape[1]} columns')
```

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

In [437…
```
#Displaying how many rows and columns using the f string
print(f'the dataset has {df_1.shape[0]} rows and {df_1.shape[1]} columns')
```

```
the dataset has 62 rows and 2 columns
```

In [438…
```
# describe for descriptive statistics
df.describe()
```

Out[438]:

	Number.of.Engines	Total.Fatal.Injuries	Total.Serious.Injuries	Total.Minor.Injuries	Total.Uninj
count	82805	77488	76379	76956	8
mean	1	1	0	0	
std	0	5	2	2	
min	0	0	0	0	
25%	1	0	0	0	
50%	1	0	0	0	
75%	1	0	0	0	
max	8	349	161	380	

In [439…
```
# describe for descriptive statistics
df_1.describe()
```

Out[439]:

	US_State	Abbreviation
count	62	62
unique	62	62
top	Alabama	AL
freq	1	1

In [440…

```
# info for overview of the data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 88889 entries, 0 to 88888
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Event.Id                88889 non-null  object 
 1   Investigation.Type      88889 non-null  object 
 2   Accident.Number         88889 non-null  object 
 3   Event.Date              88889 non-null  object 
 4   Location                88837 non-null  object 
 5   Country                 88663 non-null  object 
 6   Latitude                34382 non-null  object 
 7   Longitude               34373 non-null  object 
 8   Airport.Code            50132 non-null  object 
 9   Airport.Name            52704 non-null  object 
 10  Injury.Severity         87889 non-null  object 
 11  Aircraft.damage         85695 non-null  object 
 12  Aircraft.Category       32287 non-null  object 
 13  Registration.Number     87507 non-null  object 
 14  Make                    88826 non-null  object 
 15  Model                   88797 non-null  object 
 16  Amateur.Built           88787 non-null  object 
 17  Number.of.Engines       82805 non-null  float64
 18  Engine.Type             81793 non-null  object 
 19  FAR.Description         32023 non-null  object 
 20  Schedule                12582 non-null  object 
 21  Purpose.of.flight       82697 non-null  object 
 22  Air.carrier             16648 non-null  object 
 23  Total.Fatal.Injuries    77488 non-null  float64
 24  Total.Serious.Injuries  76379 non-null  float64
 25  Total.Minor.Injuries    76956 non-null  float64
 26  Total.Uninjured         82977 non-null  float64
 27  Weather.Condition       84397 non-null  object 
 28  Broad.phase.of.flight   61724 non-null  object 
 29  Report.Status           82505 non-null  object 
 30  Publication.Date        75118 non-null  object 
dtypes: float64(5), object(26)
memory usage: 21.0+ MB
```

In [441…

```
# info for overview of the data
df_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62 entries, 0 to 61
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   US_State      62 non-null     object
 1   Abbreviation  62 non-null     object
dtypes: object(2)
memory usage: 1.1+ KB
```

In [442…
```
#checking the number of rows
len(df)
```

Out[442]:
```
88889
```

In [443…
```
#checking the number of rows
len(df_1)
```

Out[443]:
```
62
```

Data Cleaning

This is the process of removing Duplicates and unwanted observations from data.

Missing Values

There are a couple of ways to deal with missing data but it is important to note that neither is the optimal way of doing so:

> Dropping - Deleting the records with missing values. And Replacing - Updating Missing values with values, this values could be actual or approximate.

In [444…
```
#checking for missing values in df
def identify_missing_values(df):
    """Identify is the data has missing values"""
    # identify if df has missing values(df.isnull().any())
    # empty dict to store missing values
    missing = []
    for i in df.isnull().any():
        # add the bool values to empty list
        missing.append(i)
    # covert list to set (if data has missing value, the list should have true and
    missing_set = set(missing)
    if (len(missing_set) == 1):
        out = print("The Data has no missing values")
    else:
        out = print("The Data has missing values.")

    return out


identify_missing_values(df)
```

```
The Data has missing values.
```

In [445…
```
#find the total number of missing values
df.isna().sum()
```

Out[445]:

```
Event.Id                    0
Investigation.Type          0
Accident.Number             0
Event.Date                  0
Location                   52
Country                   226
Latitude                54507
Longitude               54516
Airport.Code            38757
Airport.Name            36185
Injury.Severity          1000
Aircraft.damage          3194
Aircraft.Category       56602
Registration.Number      1382
Make                       63
Model                      92
Amateur.Built             102
Number.of.Engines        6084
Engine.Type              7096
FAR.Description         56866
Schedule                76307
Purpose.of.flight        6192
Air.carrier             72241
Total.Fatal.Injuries    11401
Total.Serious.Injuries  12510
Total.Minor.Injuries    11933
Total.Uninjured          5912
Weather.Condition        4492
Broad.phase.of.flight   27165
Report.Status            6384
Publication.Date        13771
dtype: int64
```

In [446…

```
#sorting the missing values i ascending order
df.isna().sum().sort_values(ascending= False)
```

Out[446]:

```
Schedule                 76307
Air.carrier              72241
FAR.Description          56866
Aircraft.Category        56602
Longitude                54516
Latitude                 54507
Airport.Code             38757
Airport.Name             36185
Broad.phase.of.flight    27165
Publication.Date         13771
Total.Serious.Injuries   12510
Total.Minor.Injuries     11933
Total.Fatal.Injuries     11401
Engine.Type               7096
Report.Status             6384
Purpose.of.flight         6192
Number.of.Engines         6084
Total.Uninjured           5912
Weather.Condition         4492
Aircraft.damage           3194
Registration.Number       1382
Injury.Severity           1000
Country                    226
Amateur.Built              102
Model                       92
Make                        63
Location                    52
Accident.Number              0
Investigation.Type           0
Event.Id                     0
Event.Date                   0
dtype: int64
```

In [447…

```python
def missing_values(df):
    """A simple function to identify df has missing values"""
    # identify the total missing values per column
    # sort in order
    miss = df.isnull().sum().sort_values(ascending = False)

    # calculate percentage of the missing values
    percentage_miss = (df.isnull().sum() / len(df)).sort_values(ascending = False)

    # store in a dataframe
    missing = pd.DataFrame({"Missing Values": miss, "Percentage(%)": percentage_mis

    # remove values that are missing
    missing.drop(missing[missing["Percentage(%)"] == 0].index, inplace = True)

    return missing


missing_data = missing_values(df)
missing_data
```

Out[447]:

	Missing Values	Percentage(%)
Schedule	76307	1
Air.carrier	72241	1
FAR.Description	56866	1
Aircraft.Category	56602	1
Longitude	54516	1
Latitude	54507	1
Airport.Code	38757	0
Airport.Name	36185	0
Broad.phase.of.flight	27165	0
Publication.Date	13771	0
Total.Serious.Injuries	12510	0
Total.Minor.Injuries	11933	0
Total.Fatal.Injuries	11401	0
Engine.Type	7096	0
Report.Status	6384	0
Purpose.of.flight	6192	0
Number.of.Engines	6084	0
Total.Uninjured	5912	0
Weather.Condition	4492	0
Aircraft.damage	3194	0
Registration.Number	1382	0
Injury.Severity	1000	0
Country	226	0
Amateur.Built	102	0
Model	92	0
Make	63	0
Location	52	0

In [448…

```
#checking for missing values in df_1
def identify_missing_values(df_1):
    """Identify is the data has missing values"""
    # identify if df_1 has missing values(df.isnull().any())
    # empty dict to store missing values
    missing = []
    for i in df_1.isnull().any():
        # add the bool values to empty list
        missing.append(i)
    # covert list to set (if data has missing value, the list should have true and
    missing_set = set(missing)
    if (len(missing_set) == 1):
        out = print("The Data has no missing values")
    else:
```

```
        out = print("The Data has missing values.")

    return out


identify_missing_values(df_1)
```

```
The Data has no missing values
```

In [449…
```
#find the total number of missing values
df_1.isna().sum()
```

Out[449]:
```
US_State        0
Abbreviation    0
dtype: int64
```

In [450…
```
# I am going to drop columns that have roughly more than 25% of their data missing
columns_to_drop = ['Latitude', 'Longitude', 'Airport.Code', 'Airport.Name', 'Aircra
                   'Schedule', 'Air.carrier', 'Broad.phase.of.flight']

df_clean = df.drop(columns=columns_to_drop)
```

In [451…
```
#I think there are many columns which are not important for my analysis. I drop the
# and clearer analysis
more_columns_to_drop = ['Accident.Number', 'Registration.Number', 'Amateur.Built',
                   'Publication.Date', 'Publication.Date', 'Report.Status']
df_clean = df_clean.drop(columns=more_columns_to_drop)
```

In [452…
```
# I change some column titles to more managable ones
new_column_names = {'Event.Id': 'Event ID', 'Investigation.Type': ' Investigation T
                    'Aircraft.damage':'Aircraft Damage', 'Number.of.Engines':'numbe
                    'Total.Fatal.Injuries':'Total Fatal Injuries', 'Total.Serious.I
                   'Total.Minor.Injuries':' Total Minor Injuries', 'Total.Uninjured
df_clean.rename(columns=new_column_names, inplace=True)
```

In [453…
```
#checking the columns after dropping some of the columns
df_clean.columns
```

Out[453]:
```
Index(['Event ID', ' Investigation Type', 'Event Date', 'Location', 'Country',
       'Injury Severity', 'Aircraft Damage', 'Make', 'Model',
       'number Of Engines', 'Engine.Type', 'Purpose of flight',
       'Total Fatal Injuries', 'Total Serious Injuries',
       ' Total Minor Injuries', 'Total Uninjured', 'Weather Condition'],
      dtype='object')
```

In [454…
```
#checking the shape of the column
df_clean.shape
```

Out[454]:
```
(88889, 17)
```

In [455…
```
# Handeling Null values
l2=['Location','Country','Injury Severity','Model','Make']
for i in l2:
        df_clean[i].fillna(df_clean[i].mode()[0],inplace=True)
df_clean.isna().sum()
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\502441067.py:4: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assig
nment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becaus
e the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.metho
d({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perf
orm the operation inplace on the original object.


  df_clean[i].fillna(df_clean[i].mode()[0],inplace=True)
```

Out[455]:

```
Event ID                     0
 Investigation Type          0
Event Date                   0
Location                     0
Country                      0
Injury Severity              0
Aircraft Damage           3194
Make                         0
Model                        0
number Of Engines         6084
Engine.Type               7096
Purpose of flight         6192
Total Fatal Injuries     11401
Total Serious Injuries   12510
 Total Minor Injuries    11933
Total Uninjured           5912
Weather Condition         4492
dtype: int64
```

In [456…

```
# Select the columns that have data type 'object' (typically string columns)
obj_col = df_clean.select_dtypes(include='object').columns

# Iterate over each of the object-type columns
for i in obj_col:
    # Remove leading and trailing whitespace from each entry in the column
    df_clean[i] = df_clean[i].str.strip()

# Display the first 5 rows of the object-type columns to verify changes
df_clean[obj_col].head()
```

Out[456]:

	Event ID	Investigation Type	Event Date	Location	Country	Injury Severity	Aircraft Damage	Make	Mo
0	20001218X45444	Accident	1948-10-24	MOOSE CREEK, ID	United States	Fatal(2)	Destroyed	Stinson	10
1	20001218X45447	Accident	1962-07-19	BRIDGEPORT, CA	United States	Fatal(4)	Destroyed	Piper	PA
2	20061025X01555	Accident	1974-08-30	Saltville, VA	United States	Fatal(3)	Destroyed	Cessna	17
3	20001218X45448	Accident	1977-06-19	EUREKA, CA	United States	Fatal(2)	Destroyed	Rockwell	
4	20041105X01764	Accident	1979-08-02	Canton, OH	United States	Fatal(1)	Destroyed	Cessna	

In [457…

```
# Convert the 'Make' column to title case (capitalizing the first letter of each wo
df_clean['Make'] = df_clean['Make'].str.title()
```

```
# Remove any special characters, digits, and punctuation from the 'Make' column usi
df_clean['Make'].replace('[!@#$%^&*()_+{}|:"<>,-./?`~=;0123456789]', '', regex=True

# Get the number of unique values in the 'Make' column after converting to title ca
len(df_clean['Make'].unique())

# Remove leading and trailing whitespace from each string in the 'Make' column
df_clean['Make'] = df_clean['Make'].str.strip()

# Get the number of unique values in the 'Make' column after removing whitespace
len(df_clean['Make'].unique())
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\4280058934.py:5: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained ass
ignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becaus
e the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.metho
d({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perf
orm the operation inplace on the original object.


  df_clean['Make'].replace('[!@#$%^&*()_+{}|:"<>,-./?`~=;0123456789]', '', regex=T
rue, inplace=True)
```

Out[457]:
```
7494
```

In [458…
```
# Define a function to clean a string by splitting it on spaces and returning only
def str_clean(row):
    # Split the string into at most 2 parts using space as the delimiter
    parts = row.split(" ", 2)

    # If there is more than one part, return only the first part (before the first
    if len(parts) > 1:
        return parts[0]
    else:
        # If there is no space (only one part), return the original string
        return row

# Apply the str_clean function to every row in the 'Make' column
df_clean['Make'] = df_clean["Make"].apply(str_clean)
```

In [459…
```
# Convert all strings in the 'Model' column to uppercase
df_clean['Model'] = df_clean['Model'].apply(lambda x: x.upper())

# Replace all hyphens ('-') with spaces (' ') in the 'Model' column
df_clean['Model'] = df_clean['Model'].str.replace('-', ' ')

# Apply the str_clean function to the 'Model' column to keep only the first word or
df_clean['Model'] = df_clean['Model'].apply(str_clean)


df_clean['Model'].value_counts().sort_values(ascending=False)
```

Out[459]:
```
Model
PA              12977
152              2484
172              1842
G                1269
172N             1166
                ...
PA22/135            1
CRUISEMASTER        1
B300C               1
D140                1
CARIBOU             1
Name: count, Length: 4174, dtype: int64
```

In [460…
```
df_clean.isna().sum()
```

Out[460]:
```
Event ID                     0
 Investigation Type          0
Event Date                   0
Location                     0
Country                      0
Injury Severity              0
Aircraft Damage           3194
Make                         0
Model                        0
number Of Engines         6084
Engine.Type               7096
Purpose of flight         6192
Total Fatal Injuries     11401
Total Serious Injuries   12510
 Total Minor Injuries    11933
Total Uninjured           5912
Weather Condition          4492
dtype: int64
```

In [461…
```
# Extract the number of fatalities from the 'Injury Severity' column using regex.
# The regex pattern captures digits enclosed in parentheses.
df_clean['Fatality'] = df_clean['Injury Severity'].str.extract(r'\((\d+)\)')

# Fill missing values in the 'Fatality' column with the original 'Injury Severity'
df_clean['Fatality'].fillna(df_clean['Injury Severity'], inplace=True)

# Replace specific injury severity categories ('Non-Fatal', 'Minor', 'Serious', 'In
df_clean['Fatality'].replace({'Non-Fatal': 0, 'Minor': 0, 'Serious': 0, 'Incident':

# Update the 'Fatality' column to contain the 'Total Fatal Injuries' count if the v
# otherwise, retain the existing 'Fatality' value.
df_clean['Fatality'] = df_clean.apply(lambda row: row['Total Fatal Injuries'] if ro

# Replace 'Unavailable' with NaN to handle unavailable data points in the 'Fatality
df_clean['Fatality'].replace('Unavailable', np.nan, inplace=True)

# Convert non-NaN values in the 'Fatality' column to integers.
# This line first checks for non-NaN values and then converts them to integers.
df_clean['Fatality'][~df_clean['Fatality'].isna()] = df_clean['Fatality'][~df_clear

# Set the display format for pandas so that floating-point numbers are displayed as
pd.options.display.float_format = '{:.0f}'.format
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\235846609.py:6: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assig
nment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becaus
e the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.metho
d({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perf
orm the operation inplace on the original object.


  df_clean['Fatality'].fillna(df_clean['Injury Severity'], inplace=True)
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\235846609.py:16: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained ass
ignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becaus
e the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.metho
d({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perf
orm the operation inplace on the original object.


  df_clean['Fatality'].replace('Unavailable', np.nan, inplace=True)
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\235846609.py:20: FutureWarning:
ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain
cases, but when using Copy-on-Write (which will become the default behaviour in pa
ndas 3.0) this will never work to update the original DataFrame or Series, because
the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a
single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy

  df_clean['Fatality'][~df_clean['Fatality'].isna()] = df_clean['Fatality'][~df_cl
ean['Fatality'].isna()].astype(int)
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\235846609.py:20: SettingWithCopy
Warning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_clean['Fatality'][~df_clean['Fatality'].isna()] = df_clean['Fatality'][~df_cl
ean['Fatality'].isna()].astype(int)
```

In [462…

```
# My Fatality column with more accurate representation of Fatality counts has been
df_clean['Fatality'].value_counts()
```

Out[462]:
```
Fatality
0      70998
1       8867
2       5172
3       1588
4       1103
       ...
66         1
112        1
188        1
41         1
176        1
Name: count, Length: 125, dtype: int64
```

In [463…
```
# We can drop Fatal_injuries column now
df_clean.drop(columns=['Total Fatal Injuries'], inplace=True)
```

In [464…
```
# changing date type to the appropriate format and creating a column for seasons
df_clean['Event Date'] = pd.to_datetime(df_clean['Event Date'], format='%Y-%m-%d')
df_clean['Month'] = df_clean['Event Date'].dt.month
# I am going to create more new columns out of Date columns for future analysis and
df_clean['Year'] = df_clean['Event Date'].dt.year
```

In [465…
```
Aviation_data = Aviation_data = df_clean[df_clean['Country'] == 'United States']
Aviation_data.reset_index(drop=True, inplace=True)
```

In [466…
```
# Retrieve valid state abbreviations from the 'Abbreviation' column of df_1.
valid_state_codes = df_1['Abbreviation']

# Define a function to extract the city and state from the 'Location' column.
def extract_city_state(location):
    # Check if 'location' is not NaN (i.e., valid data).
    if pd.notna(location):
        # Strip leading/trailing whitespace from the 'location' string.
        location = location.strip()

        # Extract the last two characters from the 'location' string and convert th
        last_two_chars = location[-2:].upper()

        # Check if the last two characters match a valid state abbreviation.
        if last_two_chars in valid_state_codes:
            # If valid, return the city (all characters except the last three) and
            return location[:-3].strip(), last_two_chars
        else:
            # If no valid state abbreviation is found, return the entire location a
            return location, "Not Applicable"  # Some accidents may not have occurr
    else:
        # If the 'location' is NaN, return NaN for both city and state.
        return np.nan, np.nan

# Apply the 'extract_city_state' function to the 'Location' column of the 'Aviation
# The function returns a tuple, so use .apply(pd.Series) to split the tuple into tw
Aviation_data[['City', 'State']] = Aviation_data['Location'].apply(extract_city_sta

# Ensuring that the city names are stripped of any extra whitespace after splitting
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\648305920.py:27: SettingWithCopy
Warning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  Aviation_data[['City', 'State']] = Aviation_data['Location'].apply(extract_city_
state).apply(pd.Series)
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\648305920.py:27: SettingWithCopy
Warning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  Aviation_data[['City', 'State']] = Aviation_data['Location'].apply(extract_city_
state).apply(pd.Series)
```

In [467…

```
# getting rid of trailing commas in City column
Aviation_data['City'] = Aviation_data['City'].str.rstrip(',')
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\4209244403.py:2: SettingWithCopy
Warning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  Aviation_data['City'] = Aviation_data['City'].str.rstrip(',')
```

In [468…

```
# having devided Location and Date columns, now we can drop these as well
Aviation_data.drop(columns=['Event Date', 'Location'], inplace=True)
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\2400064730.py:2: SettingWithCopy
Warning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  Aviation_data.drop(columns=['Event Date', 'Location'], inplace=True)
```

In [469…

```
# I noticed Fatality column has somehow returned to object type (not sure why?) so
Aviation_data['Fatality'] = pd.to_numeric(Aviation_data['Fatality'], errors='coerce
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\3530930609.py:2: SettingWithCopy
Warning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  Aviation_data['Fatality'] = pd.to_numeric(Aviation_data['Fatality'], errors='coe
rce')
```

In [470…

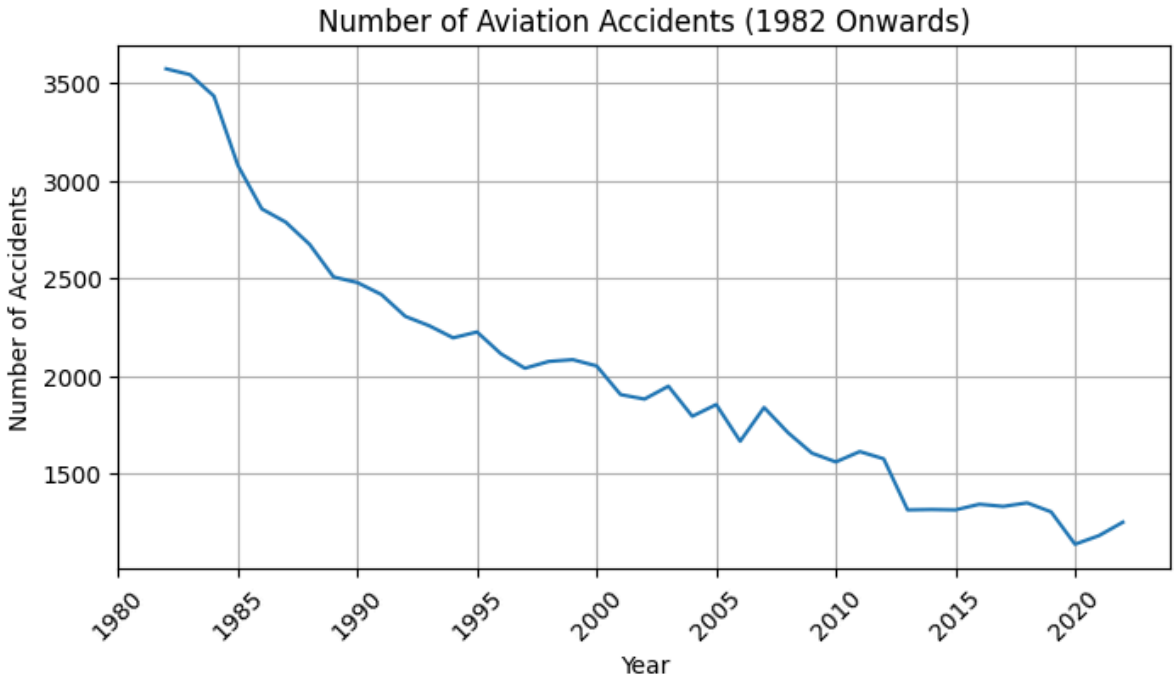
```
# Now my dataframe is ready for further analysis. There are still some missing data
# significant to impact my analysis
Aviation_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 82474 entries, 0 to 82473
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Event ID                82474 non-null  object 
 1    Investigation Type     82474 non-null  object 
 2   Country                 82474 non-null  object 
 3   Injury Severity         82474 non-null  object 
 4   Aircraft Damage         80479 non-null  object 
 5   Make                    82474 non-null  object 
 6   Model                   82474 non-null  object 
 7   number Of Engines       80591 non-null  float64
 8   Engine.Type             79426 non-null  object 
 9   Purpose of flight       80038 non-null  object 
 10  Total Serious Injuries  71089 non-null  float64
 11   Total Minor Injuries   71735 non-null  float64
 12  Total Uninjured         77460 non-null  float64
 13  Weather Condition       81824 non-null  object 
 14  Fatality                82458 non-null  float64
 15  Month                   82474 non-null  int32  
 16  Year                    82474 non-null  int32  
 17  City                    82474 non-null  object 
 18  State                   82474 non-null  object 
dtypes: float64(5), int32(2), object(12)
memory usage: 11.3+ MB
```

Data Visualization

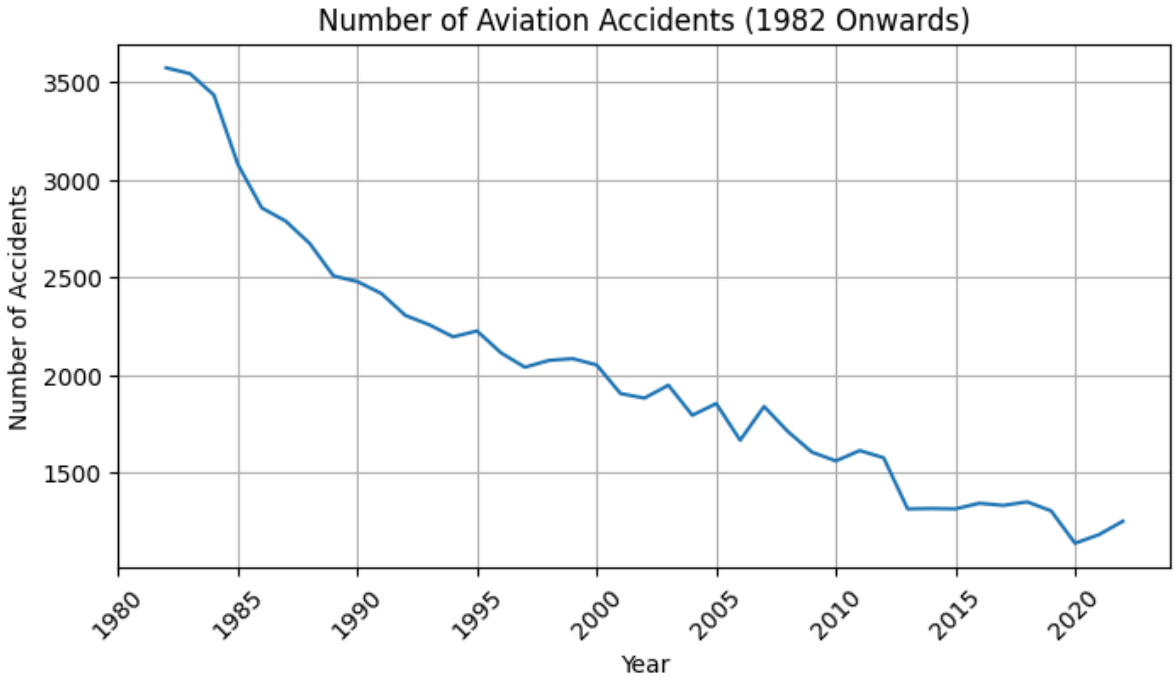
In [471…

```
# Let's have a look at the line plot of US aviation accidents over years. I noticed
# data about years before 1982. So I limited the plot from 1982 onwards

df_us_filtered = Aviation_data[Aviation_data['Year'] >= 1982]

accidents_by_year = df_us_filtered['Year'].value_counts().sort_index()

plt.figure(figsize=(8, 4))
plt.plot(accidents_by_year.index, accidents_by_year.values, linestyle='-')
plt.title('Number of Aviation Accidents (1982 Onwards)')
plt.xlabel('Year')
plt.ylabel('Number of Accidents')
plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```



In [472…

```
# Let's have a look at the line plot of US aviation accidents over years. I noticed
# data about years before 1982. So I limited the plot from 1982 onwards

df_us_filtered = Aviation_data[Aviation_data['Year'] >= 1982]

accidents_by_year = df_us_filtered['Year'].value_counts().sort_index()

plt.figure(figsize=(8, 4))
plt.plot(accidents_by_year.index, accidents_by_year.values, linestyle='-')
plt.title('Number of Aviation Accidents (1982 Onwards)')
plt.xlabel('Year')
plt.ylabel('Number of Accidents')
plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```



Observation

> The graph above shows us that the number of accidents were very high starting from 1980 and then it drops by as the years pass on and somehow increases a little bit around 1995 and keeps on dropping and increasing a liitle bit until it somehiw also drops around 2020 i think because of the coronavirus and increases after 2020.

In [473…

```
# Exploring the rate of fatalities to see what year had the most percentage of dead

# Create a copy of the original dataset to preserve the original data
df_us_filtered = Aviation_data.copy()

# Filter the dataset to only include accidents from the year 1982 onwards
# This is likely because there were only a few accidents before 1982
df_us_filtered = df_us_filtered[df_us_filtered['Year'] >= 1982]

# Group the filtered data by 'Year' and aggregate two values:
# 'count' - the number of accidents in each year
# 'sum' - the total number of fatalities in each year
yearly_stats_filtered = df_us_filtered.groupby('Year')['Fatality'].agg(['count', 's

# Calculate the fatality percentage for each year
# (total fatalities in a year / total accidents in that year) * 100
yearly_stats_filtered['Fatality_Percentage'] = (yearly_stats_filtered['sum'] / year

# Fill any NaN values (e.g., where there were no accidents or no fatalities) with n
# This is done to handle cases where there may be no fatalities for the calculation
yearly_stats_filtered['Fatality_Percentage'].fillna(-np.inf, inplace=True)

# Identify the year with the highest fatality percentage
year_with_highest_percentage = yearly_stats_filtered['Fatality_Percentage'].idxmax(

# Get the value of the highest fatality percentage
highest_percentage = yearly_stats_filtered['Fatality_Percentage'].max()

# Print the year with the highest percentage of fatalities and the percentage value
print(f"Year with the highest percentage of fatalities (1982 onwards): {year_with_h
print(f"Highest percentage of fatalities (1982 onwards): {highest_percentage:.2f}%"
```

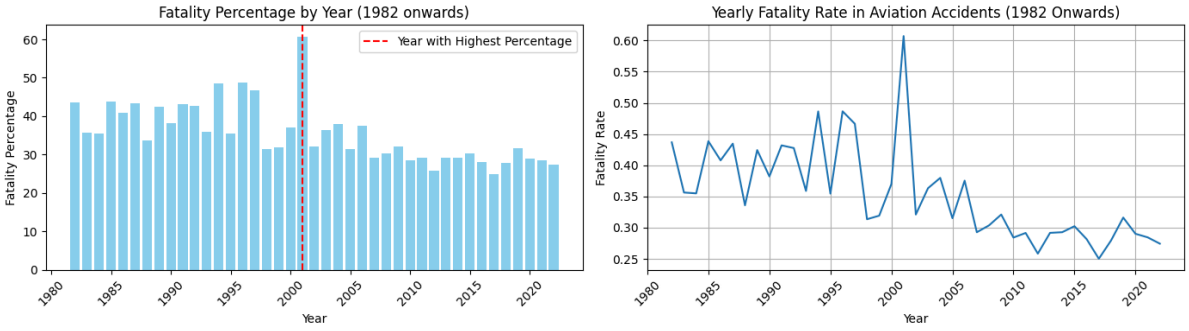
```
Year with the highest percentage of fatalities (1982 onwards): 2001
Highest percentage of fatalities (1982 onwards): 60.71%
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\3166937927.py:21: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained ass
ignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becaus
e the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.metho
d({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perf
orm the operation inplace on the original object.


  yearly_stats_filtered['Fatality_Percentage'].fillna(-np.inf, inplace=True)
```

In [474…

```
# Here I visualise above calculation

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 4))


ax1.bar(yearly_stats_filtered.index, yearly_stats_filtered['Fatality_Percentage'],
ax1.set_xlabel('Year')
ax1.set_ylabel('Fatality Percentage')
```

```
ax1.set_title('Fatality Percentage by Year (1982 onwards)')
ax1.axvline(x=year_with_highest_percentage, color='red', linestyle='--', label='Yea
ax1.legend()
yearly_fatality_rate = df_us_filtered.groupby('Year')['Fatality'].sum() / df_us_fil
ax2.plot(yearly_fatality_rate.index, yearly_fatality_rate, linestyle='-')
ax2.set_title('Yearly Fatality Rate in Aviation Accidents (1982 Onwards)')
ax2.set_xlabel('Year')
ax2.set_ylabel('Fatality Rate')
ax2.grid(True)


ax1.tick_params(axis='x', rotation=45)
ax2.tick_params(axis='x', rotation=45)
plt.tight_layout()


plt.show()
```



Observation

> The above graph shows us that the Year with the highest percentage of fatalities 1982 onwards was 2001 and the Highest percentage of fatalities 1982 onwards 60.76%.

In [475…

```
# I want to create a new column to catagorize injury types and see what portion of
# loss of life or serious injury
```

In [476…

```
# Function to categorize the severity of injuries based on the 'Injury Severity' co
def categorize_injury_severity(severity):
    # If the severity is NaN (missing), return it as is
    if pd.isna(severity):
        return severity

    # If the severity contains "Non-Fatal", categorize it as "Non-Fatal"
    elif "Non-Fatal" in severity:
        return "Non-Fatal"

    # If the severity contains either "Fatal" or "Serious", categorize it as "Fatal
    elif "Fatal" in severity or "Serious" in severity:
        return "Fatal/Serious"

    # If the severity contains "Minor" or "Incident", categorize it as "Minor"
    elif "Minor" in severity or "Incident" in severity:
        return "Minor"

    # If none of the above conditions are met, return the original severity value
    else:
        return severity

# Apply the 'categorize_injury_severity' function to the 'Injury Severity' column c
```

```
# This creates a new column 'Category' with the categorized values
Aviation_data['Category'] = Aviation_data['Injury Severity'].apply(categorize_injur
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\3173243178.py:25: SettingWithCop
yWarning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  Aviation_data['Category'] = Aviation_data['Injury Severity'].apply(categorize_in
jury_severity)
```

In [477…

```
# Replace occurrences of the string "Unavailable" in the 'Category' column with NaN
# This helps to standardize the handling of missing or uninformative data
Aviation_data['Category'] = Aviation_data['Category'].replace("Unavailable", np.nar
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\2816217911.py:3: SettingWithCopy
Warning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  Aviation_data['Category'] = Aviation_data['Category'].replace("Unavailable", np.
nan)
```

In [478…

```
# I put a more suitable name for this column
Aviation_data.rename(columns={'Category': 'Severity_Category'}, inplace=True)
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\4288991073.py:2: SettingWithCopy
Warning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  Aviation_data.rename(columns={'Category': 'Severity_Category'}, inplace=True)
```

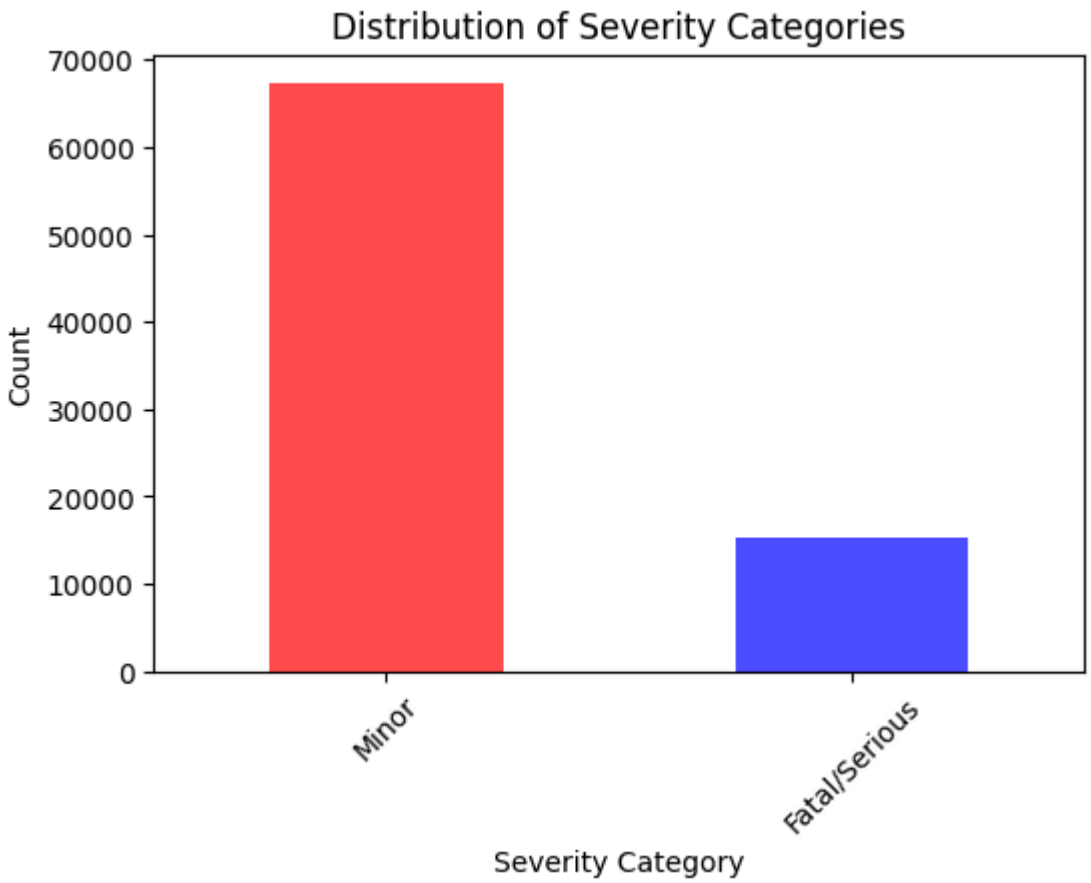
In [479…

```
# I want to have only 2 categories in this column
Aviation_data['Severity_Category'] = Aviation_data['Severity_Category'].replace("No
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\3070947602.py:2: SettingWithCopy
Warning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  Aviation_data['Severity_Category'] = Aviation_data['Severity_Category'].replace
("Non-Fatal", "Minor")
```

In [480…

```
severity_counts = Aviation_data['Severity_Category'].value_counts()


plt.figure(figsize=(6, 4))
severity_counts.plot(kind='bar', color=['red', 'blue'], alpha=0.7)
plt.title('Distribution of Severity Categories')
plt.xlabel('Severity Category')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



Observation

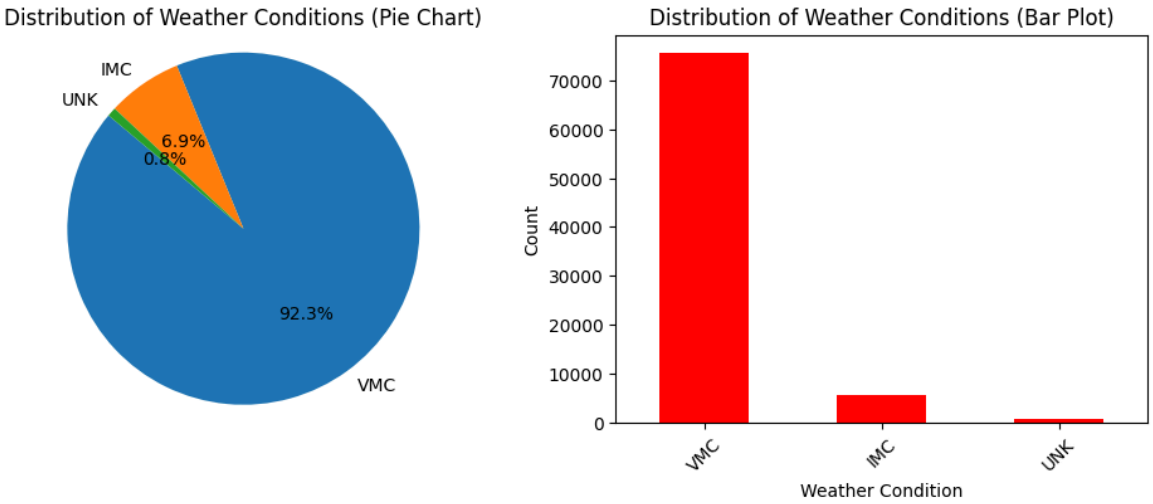
> As the bar graph above shows Most accidents were Non_Fatal or resulted in Minor injuries

In [481…

```
# I would like to explore if there is a meaningful relationship between wheather co

# in Weather column'Unknown' value was writen in both upper and lower cases and it
# Convert all values in the 'Weather Condition' column to uppercase to ensure unifo
Aviation_data['Weather Condition'] = Aviation_data['Weather Condition'].str.upper()

# Count the frequency of each unique weather condition after standardizing the text
weather_counts_updated = Aviation_data['Weather Condition'].value_counts()

# Create two subplots (side by side) with a figure size of 12x4 inches
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Plot a pie chart using the weather condition counts on the first subplot
axes[0].pie(weather_counts_updated,
            labels=weather_counts_updated.index,  # Set the labels to the weather c
            autopct='%1.1f%%',                    # Display percentage on the pie s
            startangle=140)                       # Start the pie chart at 140 degr
axes[0].set_title('Distribution of Weather Conditions (Pie Chart)')  # Set the titl
axes[0].axis('equal')   # Ensure the pie chart is drawn as a circle (equal aspect r

# Plot a bar chart using the weather condition counts on the second subplot
weather_counts_updated.plot(kind='bar', color='red', ax=axes[1])
axes[1].set_title('Distribution of Weather Conditions (Bar Plot)')  # Set the title
axes[1].set_xlabel('Weather Condition')  # Label the x-axis
axes[1].set_ylabel('Count')              # Label the y-axis
axes[1].tick_params(axis='x', rotation=45)  # Rotate the x-axis labels for better r
```

```
# Show the complete plot with both subplots
plt.show()
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\624778882.py:5: SettingWithCopyW
arning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  Aviation_data['Weather Condition'] = Aviation_data['Weather Condition'].str.uppe
r()
```



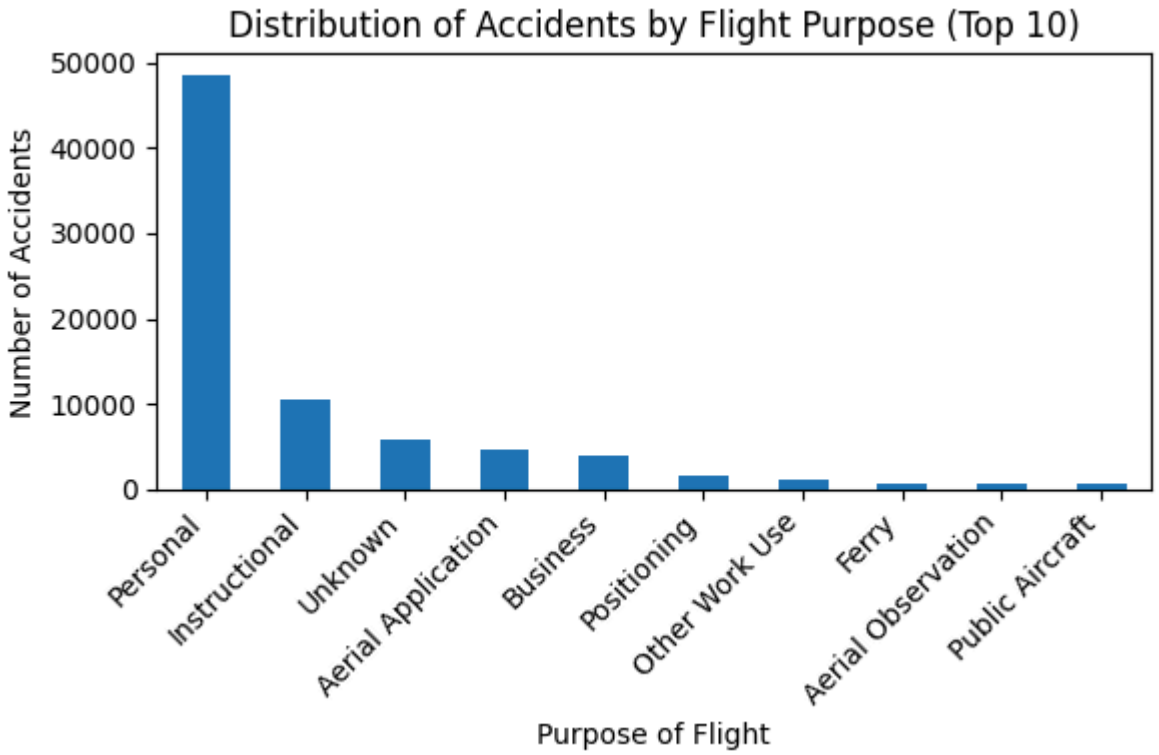
Observation

> Counterintuitively most accidents happen in (VMC) which is more favourable condition for pilots and flights. This could be due to a high number of flights when the weather is favourable.

In [482…

```
#Exploring the purpose of flights involved in accidents

# Count the occurrences of each flight purpose and extract the top 10 most frequent
top_10_purposes = Aviation_data['Purpose of flight'].value_counts().nlargest(10).sc

# Create a figure with specified dimensions (6 inches wide, 4 inches tall)
plt.figure(figsize=(6, 4))

# Plot the top 10 purposes of flight as a bar chart
top_10_purposes.plot(kind='bar')

# Set the title of the plot to describe what is being shown
plt.title('Distribution of Accidents by Flight Purpose (Top 10)')

# Label the x-axis as 'Purpose of Flight'
plt.xlabel('Purpose of Flight')

# Label the y-axis as 'Number of Accidents'
plt.ylabel('Number of Accidents')

# Rotate the x-axis labels by 45 degrees and align them to the right for better rea
plt.xticks(rotation=45, ha='right')

# Automatically adjust subplot parameters to fit the figure cleanly without overlap
plt.tight_layout()
```

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



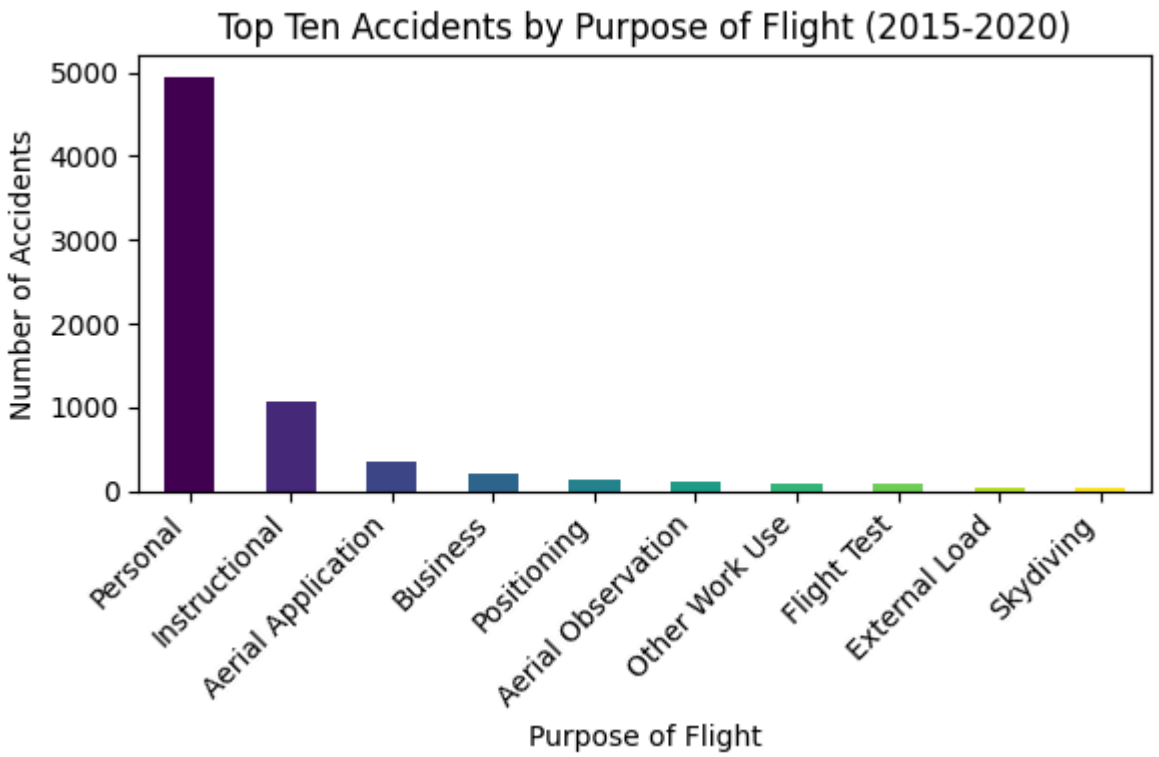
Observation

> In the bar graph above it shows that Significant number of personal flights are responsible for aviation accidents.

In [483…

```
# Filter the dataset for rows where the 'Year' is between 2015 and 2020 (inclusive)
filtered_years = Aviation_data[(Aviation_data['Year'] >= 2015) & (Aviation_data['Ye

# Count the occurrences of each flight purpose in the filtered data
purpose_counts = filtered_years['Purpose of flight'].value_counts()

# Select the top 10 most common flight purposes from the counted data
top_ten_purposes = purpose_counts.head(10)

# Generate a range of colors using the 'viridis' colormap, with colors spaced evenl
colors = plt.cm.viridis(np.linspace(0, 1, len(top_ten_purposes)))

# Set the size of the figure to 6x4 inches for readability
plt.figure(figsize=(6, 4))

# Create a bar chart with the top 10 flight purposes, using the generated color gra
top_ten_purposes.plot(kind='bar', color=colors)

# Set the title of the plot
plt.title('Top Ten Accidents by Purpose of Flight (2015-2020)')

# Label the x-axis as 'Purpose of Flight'
plt.xlabel('Purpose of Flight')

# Label the y-axis as 'Number of Accidents'
plt.ylabel('Number of Accidents')

# Rotate the x-axis labels by 45 degrees and align them to the right for better rea
plt.xticks(rotation=45, ha='right')
```

```
# Adjust the layout to ensure the plot elements fit neatly within the figure
plt.tight_layout()

# Display the final plot
plt.show()
```



Observation

> Pattern of accidents by flight purpose is similar for more recent years as well.The personal flight purpose have more accidents as they decrease until skydiving.

In [484…

```
# Trying to see the relationship between Makes, Models and engine types with accide
make_model_accident_counts = Aviation_data.groupby(['Make', 'Model']).size().reset_
make_model_accident_counts = make_model_accident_counts.sort_values(by='AccidentCou
make_model_accident_counts
```

Out[484]:

	Make	Model	AccidentCount
8064	Piper	PA	12499
2261	Cessna	152	2358
2280	Cessna	172	1716
2295	Cessna	172N	1141
4806	Grumman	G	817
...	...	...	...
21	Ac	CH	1
20	Abruzzo	GROM	1
11235	Zorn	EAA	1
11234	Zlin	Z143	1
11231	Zito	ZMI	1

11242 rows × 3 columns

In [485…
```
Aviation_data.columns
```

Out[485]:
```
Index(['Event ID', ' Investigation Type', 'Country', 'Injury Severity',
       'Aircraft Damage', 'Make', 'Model', 'number Of Engines', 'Engine.Type',
       'Purpose of flight', 'Total Serious Injuries', ' Total Minor Injuries',
       'Total Uninjured', 'Weather Condition', 'Fatality', 'Month', 'Year',
       'City', 'State', 'Severity_Category'],
      dtype='object')
```

In [486…
```
Aviation_data['Engine.Type'].value_counts()
```

Out[486]:
```
Engine.Type
Reciprocating    68617
Turbo Shaft       3416
Turbo Prop        3217
Turbo Fan         2101
Unknown           1390
Turbo Jet          669
Electric            10
LR                   2
NONE                 2
Hybrid Rocket        1
UNK                  1
Name: count, dtype: int64
```

In [487…
```
#Filling the column of Engine type with median to eliminate the null values
Aviation_data['Engine.Type'].fillna(value='median',inplace=True)
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\2116772405.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained ass
ignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becaus
e the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.metho
d({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perf
orm the operation inplace on the original object.


  Aviation_data['Engine.Type'].fillna(value='median',inplace=True)
C:\Users\lenovo\AppData\Local\Temp\ipykernel_6200\2116772405.py:2: SettingWithCopy
Warning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  Aviation_data['Engine.Type'].fillna(value='median',inplace=True)
```

In [488…

```
Aviation_data['number Of Engines'].value_counts()
```

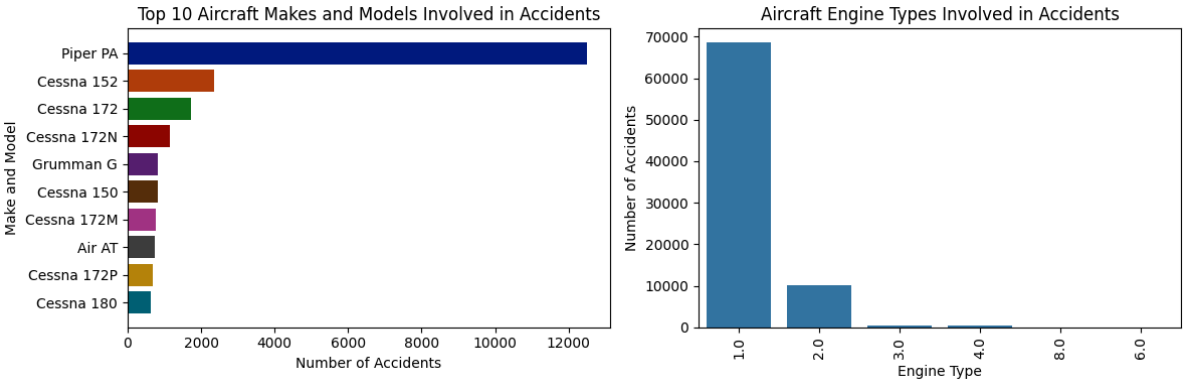
Out[488]:

```
number Of Engines
1    68542
2    10158
0     1109
3      437
4      341
8        3
6        1
Name: count, dtype: int64
```

In [489…

```
# Safely replace 0.0 with np.nan in the 'number Of Engines' column
Aviation_data.loc[Aviation_data['number Of Engines'] == 0.0, 'number Of Engines'] =

# Create a subplot with two charts
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Top 10 aircraft make and model involved in accidents
top_10_make_model = make_model_accident_counts.head(10)
colors = sns.color_palette("dark", len(top_10_make_model))

# Plot a horizontal bar chart for the top 10 makes and models
axes[0].barh(top_10_make_model['Make'] + ' ' + top_10_make_model['Model'],
             top_10_make_model['AccidentCount'], color=colors)
axes[0].set_xlabel('Number of Accidents')
axes[0].set_ylabel('Make and Model')
axes[0].set_title('Top 10 Aircraft Makes and Models Involved in Accidents')
axes[0].invert_yaxis()

# Plot a countplot for the number of engines if data exists
if not Aviation_data['number Of Engines'].isnull().all():
    sns.countplot(data=Aviation_data,
                  x='number Of Engines',
                  order=Aviation_data['number Of Engines'].value_counts().index,
                  ax=axes[1])
    axes[1].set_title('Aircraft Engine Types Involved in Accidents')
    axes[1].set_ylabel('Number of Accidents')
    axes[1].set_xlabel('Engine Type')
    axes[1].tick_params(axis='x', rotation=90)
else:
    axes[1].set_title('No Engine Data Available')

# Display the plot
```

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

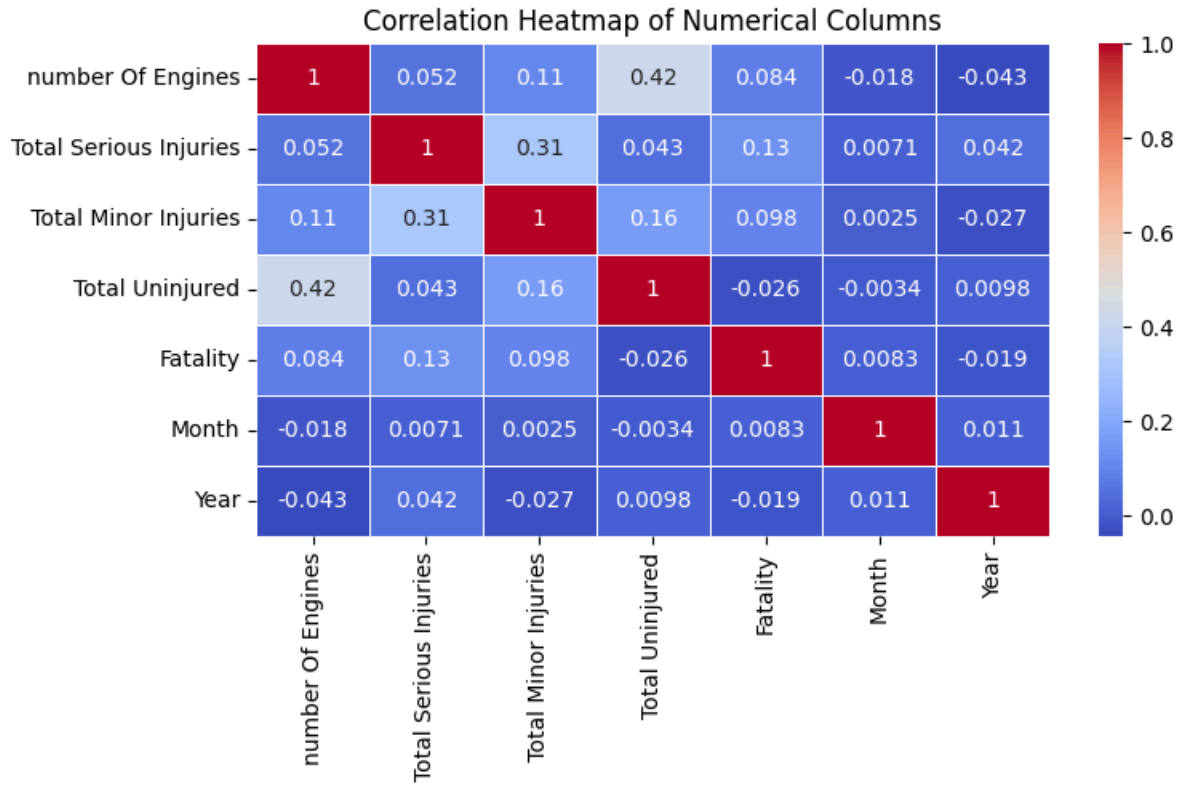


Observation

> Cessna152, Cessna172, Cessna172N invoved in more accidents. This might be due to the populariy of these models but more investigation must be conducted. There also seems to be a strong connection between engine type and number of accidents. But again, this can be because of high number of them being in operation.

In [490…

```
# A correlation heatmap of numerical values
numerical_data = Aviation_data.select_dtypes(include=['number'])

correlation_matrix = numerical_data.corr()

plt.figure(figsize=(8, 4))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap of Numerical Columns')
plt.show()
```



Observation

> There are mostly strong negative and occasionally weak negative correlation between numerical columns.

> The bottom line for the "success" of a the company purchasing Aircrafts with the lowest risk is going with the aircraft involved with the least accidents Variables that may influence the "success" of a Safe Aircraft:

1. Make
2. Model
3. weather condition
4. Engine type
5. Injury severity