Business Understanding

Background:

My company is planning to diversify its porfolio by extending to different markets. More specifically, my company is looking to invest in airplanes in order for their expansion to different markets to be successful.

Business goals:

The primary focus of this data science project is to assess and mitigate the risks associated with various aircraft models. Specifically, we aim to identify and recommend aircraft types that carry the least potential risk. This information is crucial for making informed decisions regarding aircraft investments and ensuring the success of our company's expansion into new markets.

Business success criteria:

The success of this project will be measured by providing three well-supported recommendations on the least risky aircraft models for our company to invest in. For this project, the term least risky refers to types of aircrafts with the least amount of crashes, the least number of casualties, and the lowest fatality rate.

Data Understanding

The National Transportation Safety Board (NTSB) collects data on aviation accidents and incidents that occur in the United States (which include its territories) as well as international waters. This dataset includes 31 features and 88,889 observations or entries.

Each entry in the dataset represents an aircraft involved in an accident (or incident). For each aircraft there is a unique ID associated with the specific accident (or incident) the aircraft was involved in. Additional information is included about each entry, such as the accident (or incident) date, location, and number of injuries, as well as characteristics about the aircraft, such as the make, model, and number of engines.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
df = pd.read_csv('AviationData.csv', encoding = 'latin-1')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-2-f15b65f0fd27> in <module>
----> 1 df = pd.read_csv('AviationData.csv', encoding = 'latin-1')

~/anaconda3/envs/learn-env/lib/python3.8/site-packages/pandas/io/parsers.py in read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters, true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates, iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding, dialect, error_bad_lines, warn_bad_lines, delim_whitespace, low_memory, memory_map, float_precision)
    684     )
    685
--> 686     return _read(filepath_or_buffer, kwds)
    687
    688

~/anaconda3/envs/learn-env/lib/python3.8/site-packages/pandas/io/parsers.py in _read(filepath_or_buffer, kwds)
    450
    451     # Create the parser.
--> 452     parser = TextFileReader(fp_or_buf, **kwds)
    453
    454     if chunksize or iterator:

~/anaconda3/envs/learn-env/lib/python3.8/site-packages/pandas/io/parsers.py in __init__(self, f, engine, **kwds)
    944             self.options["has_index_names"] = kwds["has_index_names"]
    945
--> 946         self._make_engine(self.engine)
    947
    948     def close(self):

~/anaconda3/envs/learn-env/lib/python3.8/site-packages/pandas/io/parsers.py in _make_engine(self, engine)
   1176     def _make_engine(self, engine="c"):
   1177         if engine == "c":
-> 1178             self._engine = CParserWrapper(self.f, **self.options)
   1179         else:
   1180             if engine == "python":

~/anaconda3/envs/learn-env/lib/python3.8/site-packages/pandas/io/parsers.py in __init__(self, src, **kwds)
   1989         if kwds.get("compression") is None and encoding:
   1990             if isinstance(src, str):
-> 1991                 src = open(src, "rb")
   1992                 self.handles.append(src)
   1993
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'AviationData.csv'
```

In [ ]:
```
df.head()
```

In [ ]:
```
df.shape
```

In [ ]:
```
df.info()
```

In [ ]:
```
# Summary statistics for features of float64 type
df.describe()
```

Data Preparation

In data preparation, I created three functions (`display_df_information()` , `multiple_value_counts()` , and `examine_features()`) to assess data types, NaN values, and entry details. Columns with high NaN values or high cardinality with a uniform distribution were dropped: `Longitude` , `Latitude` , `FAR.Description` , `Schedule` , `Air.carrier` , `Airport.Code` , `Airport.Name` , `Publication.Date` , `Report.Status` , `Registration.Number` , `Investigation.Type` , `Purpose.of.flight` .

Focused on professionally built airplanes, I filtered rows where `Aircraft.Type` was 'airplane' and `Professionally.Built` was 'yes'. `Injury.severity` values were grouped into 'Non-Fatal', 'Fatal', and 'Unavailable', consolidating data from `Total.Fatal.Injuries` .

Using a `lambda` function, I standardized case sensitivity and removed duplicates in the `Make` column. Rows with a small percentage of NaN values were dropped using `.dropna()` .

Remaining NaN values in `Total.Fatal.Injuries` , `Total.Serious.Injuries` , `Total.Minor.Injuries` , and `Total.Uninjured` were retained due to their significant count, avoiding noise addition. The cleaned dataframe now has 18 columns and over 65,000 entries.

In [ ]:
```
#function that displays the dataframe .info() and .isna().sum()
def display_df_information(dataframe):
    print(dataframe.info())
    print()
    print('--- Number of NaNs per Column ---')
    print(dataframe.isna().sum())
    print()
```

In [ ]:
```
display_df_information(df)
```

In [ ]:
```
#function uses a for loop to print the column name and .value_counts() fo
def multiple_value_counts(dataframe,list_of_columns):
    for col in list_of_columns:
        print("----" + col + "----")
        print()
        print(dataframe[col].value_counts())
        print()
        print("There are " + str(dataframe[col].nunique()) + ' unique en
        print()
```

In [ ]:
```
#Prints the .value_counts() for all the columns of the dataframe
multiple_value_counts(df, df.columns)
```

In [ ]:
```
#function that examines features with more closely
#by printing the first ten entries and data types for each of these featu

def examine_features(dataframe, list_of_features):
    if type(list_of_features) == list:
        for feature in list_of_features:
            print("---" + feature + ' First 10 entries' + '---')
            print(dataframe[feature].head(10))
            print()
            print(print("---" + feature + ' First 10 data types' + '---'
            for entry in dataframe[feature][:10]:
                print(type(entry))
            print('\n')
    else:
        print("---" + list_of_features + ' First 10 entries' + '---')
        print(dataframe[list_of_features].head(10))
        print()
        print(print("---" + list_of_features + ' First 10 data types' +
        for entry in dataframe[list_of_features][:10]:
            print(type(entry))
        print('\n')
```

In [ ]:
```
cols_over_500_nan = list(df.columns[df.isna().sum() > 500])
```

In [ ]:
```
examine_features(df, cols_over_500_nan)
```

In [ ]:
```
# To make for easier handling of my data, I remove the following columns
# large number of missing values or they do not provide information that

cols_to_drop = ['Longitude', 'Latitude', 'FAR.Description', 'Schedule',
                'Airport.Name', 'Publication.Date', 'Report.Status', 'Re
                'Investigation.Type', 'Purpose.of.flight']
```

In [ ]:
```
#Removes columns in cols_to_drop
df_clean = df.drop(cols_to_drop, axis = 1)
display_df_information(df_clean)
```

In [ ]:
```
#Creates a list of non-airplane aircraft categories
non_airplanes = list(df['Aircraft.Category'].value_counts()[1:].index)
non_airplanes
```

In [ ]:
```
#uses lambda function to filter out any non-airplane labeled aircraft us
df_clean = df_clean.apply(lambda row: row[~df_clean['Aircraft.Category']
df_clean['Aircraft.Category'].value_counts()
```

In [ ]:
```
#Drops the Aircraft.Category column since it only contains airplanes
df_clean = df_clean.drop(['Aircraft.Category'], axis = 1)
df_clean.info()
```

In [ ]:
```
df_clean['Amateur.Built'].value_counts()
```

In [ ]:
```
#Creates a new dataframe consisting only of professionally built airplan
df_clean = df_clean[(df_clean['Amateur.Built'] == 'No')]
df_clean['Amateur.Built'].value_counts()
```

In [ ]:
```
#Drops the Amateur.Built column since now all data in this column is onl
df_clean = df_clean.drop(['Amateur.Built'], axis = 1)
display_df_information(df_clean)
```

In [ ]:
```
#For Injury.Severity, group all fatal injuries into the same category na
#specifying the number of fatalities since the Total.Fatal.Injuries colu

df_clean['Injury.Severity'] = df_clean['Injury.Severity'].map(lambda ent
df_clean['Injury.Severity'].value_counts()
```

In [ ]:
```
#Grouping minor injuries into the Non-Fatal category
df_clean['Injury.Severity'].replace({'Minor': 'Non-Fatal'}, inplace = Tr
df_clean['Injury.Severity'].replace({'Serious': 'Non-Fatal'}, inplace = 
df_clean['Injury.Severity'].value_counts()
```

In [ ]:
```
#Grouping nan values into the Unavailable category
df_clean['Injury.Severity'].replace({'nan': 'Unavailable'}, inplace = Tr
df_clean['Injury.Severity'].value_counts()
```

In [ ]:
```
#Removes all rows where Injury.Severity is 'Incident'
df_clean = df_clean[df_clean['Injury.Severity'] != 'Incident']
df_clean['Injury.Severity'].value_counts()
```

In [ ]:
```
df_clean.isna().sum()
```

In [ ]:
```
#Grouping UNK values into the Unknown category
df_clean['Engine.Type'].replace({'UNK': 'Unknown'}, inplace = True)
df_clean['Engine.Type'].value_counts()
```

In [ ]:
```
#Converting all entries in Engine.Type to string data type
df_clean['Engine.Type'] = df_clean['Engine.Type'].astype(str)
```

In [ ]:
```
examine_features(df_clean, ['Engine.Type'])
```

In [ ]:
```
display_df_information(df_clean)
```

In [ ]:
```
df_clean = df_clean[df_clean['Number.of.Engines'].notna()]
display_df_information(df_clean)
```

In [ ]:
```
df_clean['Make'][3]
```

In [ ]:
```
df_clean['Make'] = df_clean['Make'].astype(str)
```

In [ ]:
```
df_clean['Make'] = df_clean['Make'].map(lambda x: x.title())
```

In [ ]:
```
examine_features(df_clean, list(df_clean.columns))
```

In [ ]:
```
multiple_value_counts(df_clean, list(df_clean.columns))
```

In [ ]:
```
df_clean = df_clean.dropna(subset = ['Weather.Condition', 'Aircraft.dama
display_df_information(df_clean)
```

In [ ]:
```
df_clean['Weather.Condition'].replace({'UNK': 'Unknown'}, inplace = True
df_clean['Weather.Condition'].replace({'Unk': 'Unknown'}, inplace = True
df_clean['Weather.Condition'].value_counts()
```

In [ ]:
```
#Removes all rows where Engine.Type is 'nan'
df_clean = df_clean[df_clean['Engine.Type'] != 'nan']
df_clean['Engine.Type'].value_counts()
```

In [ ]:
```
display_df_information(df_clean)
```

In [ ]:
```
type(df_clean['Event.Date'][2])
```

In [ ]:
```
#Convert event dates from string to pandas datetime datatype
df_clean['Event.Date'] = pd.to_datetime(df_clean['Event.Date'])
```

In [ ]:
```
type(df_clean['Event.Date'][2])
```

In [ ]:
```
#Creating a column for the day of the week for each event
df_clean['Event.Day'] = df_clean['Event.Date'].dt.day_name()
```

In [ ]:
```
display_df_information(df_clean)
```

In [ ]:
```
multiple_value_counts(df_clean, list(df_clean.columns))
```

In [ ]:
```
list_of_categorical_features_with_nan = ['Broad.phase.of.flight', 'Locat
```

In [ ]:
```
#Function that fills multiple columns' NaN values with 'Unknown'
def multiple_fill_nan(dataframe, list_of_cols):
    for col in list_of_cols:
        dataframe[col] = dataframe[col].fillna('Unknown')
```

In [ ]:
```
multiple_fill_nan(df_clean, list_of_categorical_features_with_nan)
display_df_information(df_clean)
```

In [ ]:
```
# Cleaned dataframe is saved as a csv
df_clean.to_csv('aircraft_safety_cleaned.csv')
```

Exploratory Data Analysis

The following are findings from this analysis:

- There were a total of 64,862 recorded aviation accidents since 1948
- 18.2% of accidents were classified as ‘fatal’
- Almost 90% of reported accidents involved aircraft with a Engine Type classified as 'Reciprocating'
- ~98% of reported accidents involved aircraft with either 1 or 2 engines
- Almost one-third of accidents (~31%) occurred during either the 'Landing' or 'Takeoff' phases of the flight
- Based on metrics for lowest fatality rate, least number of deaths, and lowest number of deaths per flight, the safest brands of airplanes include:
  - Hiller
  - Schweizer
  - Stinson
  - Maule
  - Enstrom
  - Luscombe
  - Aeronca

In [ ]:
```
# Gets the summary statistics for numerical data after dataframe is now
df_clean.describe()
```

This table tells us the summary statistics for each feature that is considered a numerical data type. This information is important because for each column, it provides the maximum and minimum values, the total count of non-null values, and measures of central tendency (mean and 50% aka median).

In [ ]:
```
#Calculates the total number of unique accidents
len(df_clean['Event.Id'].unique())
```

In [ ]:
```
# calculates the overall fatality rate across all accidents

fatal_rate = len(df_clean[df_clean['Injury.Severity'] == 'Fatal'])/len(d
print(f"Fatality rate: {round(fatal_rate*100, 2)}% ")
```

In [ ]:
```
# Calculates the percentages of the frequency of each engine type
type_of_engines_percent = round(df_clean['Engine.Type'].value_counts(nor
type_of_engines_percent
```

In [ ]:
```
# Calculate the percentages of the frequency of each number of engines
num_of_engines_percent = round(df_clean['Number.of.Engines'].value_count
num_of_engines_percent
```

In [ ]:
```
# groups data first by engine type, then by number of engines which
# allows us to see the number of fatalities and injuries by engine type
df_clean.groupby(['Engine.Type','Number.of.Engines']).sum()
```

The table above breaks down the number of injuries (fatal, serious, minor, and uninjured) based on number of engines within each engine type. This information shows that a high number of fatalities and injuries occur from aircraft with 1 or 2 engines.

In [ ]:
```
# Bar plot that visualizes the percentages of the frequency of each type
# Bar plot that visualizes the percentages of the frequency of each numbe

faulty_engines = [1.0, 2.0]
num_eng_color = ['orange' if x in faulty_engines else 'grey' for x in nu
type_eng_color = ['orange' if x == 'Reciprocating' else 'grey' for x in

fig, ax = plt.subplots(figsize = (24, 8), ncols = 2, nrows = 1)
plt.subplots_adjust(bottom=0.20)

sns.barplot(ax = ax[0],
            x = num_of_engines_percent.index,
            y = num_of_engines_percent,
            palette = num_eng_color, order = num_of_engines_percent.index

ax[0].set_title('Percents of Number of Engines', fontsize=20)
ax[0].set_xlabel('Number of Engines', fontsize=20)
ax[0].set_ylabel('Percent', fontsize=20)
ax[0].tick_params(labelsize=15)

sns.barplot(ax = ax[1],
            x = type_of_engines_percent.index,
            y = type_of_engines_percent,
            palette = type_eng_color)

ax[1].set_title('Percents of Types of Engines', fontsize=20)
ax[1].set_xlabel('Engine Type', fontsize=20)
ax[1].set_ylabel('Percent', fontsize=20)
ax[1].tick_params(labelsize=15)
plt.xticks(rotation=45)
plt.savefig("engines.jpg");
```

The above bar graph shows the percentages of each number of engines. In other words, the bars show us which number(s) of engines were involved in more accidents. Specifically, airplanes with 1 and 2 engines were involved in ~98% of reported accidents.

This bar graph shows the percentages of each category of engines. In other words, the bars show us which number(s) of engines were involved in more accidents. Specifically, airplanes with 1 and 2 engines were involved in ~98% of reported accidents.

In [ ]:
```
# creates a list of the frequencies of all the different aircraft compan
make = df_clean['Make'].value_counts()
make
```

In [ ]:
```
# creates a list of the frequencies of only the aircraft companies that
make_over_200_flights = make[make >200]
make_over_200_flights
```

In [ ]:
```
# creates a list of only the aircraft companies that had a frequencies o
top_brands = make_over_200_flights.index
top_brands
```

In [ ]:
```
# creates a dataframe from df_clean which only includes rows with the ai
df_200_flights = df_clean[df_clean['Make'].isin(top_brands)]
```

In [ ]:
```
# creates a table of the fatal and non-fatal percentages for each aircra
fatal_rate_200_flights = round(df_200_flights.groupby('Make')['Injury.Se
fatal_rate_200_flights
```

This table shows the percentages of flights for each company that were classified as 'Fatal', 'Non-Fatal', and 'Unavailable'. Together the sum of all three categories should add to 100% for each company.

In [ ]:
```
# creates a stacked bar plot with the fatal and and non-fatal percentages
# for each aircraft company in df_200_flights with
fatal_rate_200_flights.plot.bar(stacked = True, figsize = (20, 8))

plt.title('Aircraft Company Fatality and Non-Fatality Rates')
plt.xlabel('Aircraft Company')
plt.ylabel('Percent')
plt.xticks(rotation=60)
plt.axhline(y = round(fatal_rate*100, 2), color = 'black', linestyle = '
plt.text(10, round(fatal_rate*100, 2) + 2, 'Average Fatality Rate 18.2%'
plt.savefig("stacked_bar.png");
```

This stacked bar chart shows the fatal rate (blue) and non-fatal rate (orange) for each company, ordered from least fatal to most fatal. A trend line for the average fatality rate is also shown, and from this line we can see that the following companies are below the average fatality rate: Hiller, Schweizer, Stinson, Grumman, Boeing, Maule, Enstrom, Luscombe, Aeronca, Hughes, Air Tractor, Taylorcraft, Cessna, Champion, Ayres, Bell, Robinson.

In [ ]:
```
# creates a value counts of percentages of each flight phase where accid
phase_of_flight_percent = round(df_clean['Broad.phase.of.flight'].value_
phase_of_flight_percent
```

In [ ]:
```
# creates a bar graph of percentages of each flight phase where accidents
safest_phases = ['Landing', 'Takeoff']
plt.subplots(figsize = (15,8))
colors = ['orange' if x in safest_phases else 'grey' for x in phase_of_f

flight_phase_bar = sns.barplot(
    x = phase_of_flight_percent.index,
    y = phase_of_flight_percent,
    palette = colors)

flight_phase_bar.axes.set_title("Percent of Accidents by Flight Phase",fo
flight_phase_bar.set_xlabel("Flight Phase",fontsize=15)
flight_phase_bar.set_ylabel("Percent",fontsize=15)
flight_phase_bar.tick_params(labelsize=12)
plt.xticks(rotation=45)
plt.savefig("flight_phase.png");
```

This bar graph shows the portion of all accidents that occurred for each flight phase, going from the stage with the highest percentage to the lowest percentage. From this graph we see that over 20% of accidents did not have a known flight phase, and over 30% of accidents occurred during the landing or takeoff phases.

In [ ]:
```
# creates a value counts of the total number of occurrences of each airc
total_flights_per_brand = df_200_flights.groupby('Make')['Event.Id'].cou
total_flights_per_brand
```

In [ ]:
```
# creates a value counts of the total number of fatalities of each aircra
total_deaths_by_brand = df_200_flights.groupby('Make')['Total.Fatal.Inju
total_deaths_by_brand
```

In [ ]:
```
# creates a value counts of the deaths per flight rate for each aircraft
deaths_per_flight = (total_deaths_by_brand/total_flights_per_brand).sort_
deaths_per_flight
```

The following bar graph shows the amount of deaths per flight for each company, sorted from lowest deaths per flight to highest deaths per flight.

In [ ]:
```
# Creates lists of the top 10 brands based on: nonfatal rates, least numb
top_10_nonfatal_rates = fatal_rate_200_flights[:10]
top_10_least_deaths = total_deaths_by_brand[:10]
lowest_10_deaths_per_flight = deaths_per_flight[:10]
```

In [ ]:
```
# Returns a list of the brands that are in each of the three lists:
# top_10_nonfatal_rates, top_10_least_deaths, lowest_10_deaths_per_flight

top_7_safest = list(top_10_nonfatal_rates.index & top_10_least_deaths.ind
top_7_safest
```

In [ ]:
```
#creates a list of the safestest aircraft companies by non-fatal rates, 
safest_aircrafts_data = [top_10_nonfatal_rates['Non-Fatal'], top_10_leas
                         round(lowest_10_deaths_per_flight, 3)]

#turns the list into a dataframe
safest_aircrafts_df = pd.DataFrame(safest_aircrafts_data).T.dropna()

safest_aircrafts_df.rename(columns={
    'Non-Fatal': 'Non-Fatal Rate',
    'Total.Fatal.Injuries': 'Total Death Toll',
    'Unnamed 0':'Deaths per Flight'},
                           inplace=True)

safest_aircrafts_df.index.names = ['Make']
safest_aircrafts_df
```

This table shows the top 7 safest aircraft companies. These 7 companies were all in the bottom 10 of all of the following categories: fatality rates, total death toll, and deaths per flight.

In [ ]:
```
# creates 3 bar plots. One of the non-fatal rates for each aircraft compa
# and one bar plot of the total deaths toll for each aircraft company wi
# and one bar plot of the deaths per flight rate for each aircraft compa

fig, ax = plt.subplots(figsize = (22, 25), ncols = 1, nrows = 3)
fig.tight_layout(pad=17.0)

#Highlights the bars of the top 7 safest aircraft companies for each gra
non_fatal_colors = ['orange' if x in top_7_safest else 'grey' for x in fa
total_deaths_colors = ['orange' if x in top_7_safest else 'grey' for x in
deaths_per_colors = ['orange' if x in top_7_safest else 'grey' for x in

sns.barplot(ax = ax[0],
            x = fatal_rate_200_flights.index,
            y = fatal_rate_200_flights['Non-Fatal'],
            palette = non_fatal_colors)

ax[0].set_title("Non-Fatal Rates by Aircraft Company",fontsize=20)
ax[0].set_xlabel(None)
ax[0].set_ylabel("Percentage of Non-Fatal Flights",fontsize=15)
ax[0].tick_params(labelsize=15)
ax[0].set_xticklabels(labels = fatal_rate_200_flights.index, rotation=60

sns.barplot(ax = ax[1],
            x = total_deaths_by_brand.index,
            y = total_deaths_by_brand,
            palette = total_deaths_colors)

ax[1].set_title("Death Toll by Aircraft Company",fontsize=20)
ax[1].set_xlabel(None)
ax[1].set_ylabel("Total Deaths",fontsize=15)
ax[1].tick_params(labelsize=15)
ax[1].set_xticklabels(labels = total_deaths_by_brand.index, rotation=60)

sns.barplot(ax = ax[2],
            x = deaths_per_flight.index,
            y = deaths_per_flight,
            palette = deaths_per_colors)

ax[2].set_title("Deaths per Flight by Aircraft Company",fontsize=20)
ax[2].set_xlabel("Aircraft Company", fontsize = 15)
ax[2].set_ylabel("Deaths per Flight",fontsize=15)
ax[2].set_xticklabels(labels = deaths_per_flight.index,rotation=60)
ax[2].tick_params(labelsize=15);
```

This trio of bar graphs is a visualization of the data of the top 7 safest aircraft companies that was printed in the table above. The bars highlighted in orange represent the top 7 safest aircraft companies that were identified in this analysis.

- The top bar graph visualizes the percentage of flights for each company where no one died, sorted from highest to lowest percentage.
- The middle bar graph displays the total number of recorded deaths for each company, sorted from least to most.
- The bottom bar graph displays the rate of deaths per flight for each company, sorted from least to most.

Conclusions

Limitations

While this dataset offers valuable information, it only includes data on flights which were involved in an accident or incident. As a result, any flights that result in a completely safe flight are not part of this data. Because of this, the analysis was limited to simply making recommendations regarding the ***least risky*** aircrafts. In other words, the least risky aircraft might not be the safest aircraft from the pool of all possible aircrafts, but within the data we have access to we are restricted to equating the least amount of risk with being the safest aircraft. A more comprehensive dataset including data on all flights (accidents as well as no accidents) would provide a more complete analysis with more informed recommendations about the safest aircrafts.

Recommendations

This analysis leads to three recommendations for choosing the safest aircrafts:

- Invest in aircraft that have more than two engines and are not classified as a reciprocating engine.
  - Aircraft with reciprocating engines are prone to more accidents (~90%). Similarly, aircraft with one or two engines were involved in ~98% of reported accidents.
- Focus training of pilots on the Takeoff and Landing phases.
  - While there are many phases that pilots must master, accidents during the Takeoff and Landing phases account for almost one-third of all accidents (~31%).
- Select aircraft made by one of the seven following companies: Hiller, Schweizer, Stinson, Maule, Enstrom, Luscombe, Aeronca
  - Planes made by these aircraft manufacturers were found to hold the least amount of risk based on a combination of their non-fatal rate, deaths per flight, and overall number of deaths.

Next Steps

With these recommendations in mind, for next steps I would be interested in gathering and looking into two sets of data:

- price data for aircraft from the recommended companies, as the price will be a major factor in deciding which aircraft(s) to invest in
- pilot training program data to help make an informed decision for where to hire the pilots from

Additional Resources

- GitHub Repository (https://github.com/ckucewicz/aircraft_safety_project)
- Tableau Dashboard (https://public.tableau.com/views/AircraftSafetyDashboard/Dashboard2?:language=en-US&publish=yes&:display_count=n&:origin=viz_share_link)
- Slideshow Presentation (https://github.com/ckucewicz/aircraft_safety_project/blob/master/Phase%201%20Final%20

Works Cited:

- When creating visualizations in my data analysis I ran into the problem of highlighting specific bars of a bar graph in order to make them stand out. Specifically, I wanted the bars of specific aircraft companies to be a different color in order to emphasize their safety compared to other aircraft companies. I googled 'highlight bars of a bar graph in matplotlib and found a solution on stackoverflow (https://stackoverflow.com/questions/52931912/highlight-particular-bars-in-bar-graph) written by Diziet Asahi which used the following code:

```
fig, ax = plt.subplots()
  c = ['C2' if i in df2.index else 'C1' for i in df1.index]
  ax.barh(y=df1.index,width=df1.col1,color=c)
  ax.grid(False)
```

This solution helped me to create bar graphs with certain bars having a different color to differentiate them from the rest.

- When creating a stacked bar graph of the fatality and non-fatal rates of the top 200 aircraft companies, I ran into the problem of including a trend line to show the average fatality rate of all companies. I googled 'show horizontal line on bar graph in matplotlib' and found a solution on stackoverflow (https://stackoverflow.com/questions/43560801/display-y-axis-value-horizontal-line-drawn-in-bar-chart) written by ImportanceOfBeingErnest which used the following code:

```
fig = plt.figure()
   ax = fig.add_subplot(111)
   x = [0,1,2,3]
   y = np.array([34,40,38,50])*1e3
   norm = matplotlib.colors.Normalize(30e3, 60e3)
   ax.bar(x,y, color=plt.cm.plasma_r(norm(y)) )
   ax.axhline(4.2e4, color="gray")
   ax.text(1.02, 4.2e4, "42000", va='center', ha="left",
bbox=dict(facecolor="w",alpha=0.5),
   transform=ax.get_yaxis_transform())
   plt.show()
```

This solution helped me to create a trend line on my stacked bar graph plot in order to show