For a project, I determined I needed more data on the planes so I could determine the likelihood of mechanical errors, delays, or other issues.

I could not find a robot.txt for FAA.gov, but based on my research I determined it was permissable.

I had plenty of time to pull the data, so I put in a long sleep time to try and not overload the website.

If this is against any rules or regulations, I'm very sorry, please contact me and I will take this down. Given this was a school project (nonprofit), I surmised this was acceptable, especially after looking at the following.

https://www.pilotsofamerica.com/community/threads/api-for-faa-aircraft-database.125423/

https://blog.exploratory.io/scrape-data-from-web-pages-50e45b2b150a

In [ ]:
```
#import will2live
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.firefox.service import Service as FirefoxService
from selenium.webdriver.firefox.options import Options
from bs4 import BeautifulSoup
import pandas as pd
import numpy as np
import time
from selenium.common.exceptions import WebDriverException

# Set up the path to Firefox and geckodriver
driver_path = r'C:\Users\me\dependencies\geckodriver-v0.33.0-win32\geckodriver.exe'
firefox_path = r'C:\Program Files\Mozilla Firefox\firefox.exe'

#Firefox options
firefox_options = Options()
firefox_options.binary_location = firefox_path

# disable image loading
firefox_options.set_preference('permissions.default.image', 2)
firefox_options.set_preference('dom.ipc.plugins.enabled.libflashplayer.so', 'false')
```

```
# set up Firefox service with geckodriver
firefox_service = FirefoxService(executable_path=driver_path)

# initialize the Firefox driver with the specified service and options
driver = webdriver.Firefox(service=firefox_service, options=firefox_options)

# replace with the actual URL and input_box_id
url = 'https://registry.faa.gov/aircraftinquiry/search/nnumberinquiry'
input_box_id = 'input-error'




df=pd.read_csv('file1_.csv')


# find the index of the first 'nan' occurrence in the 'Manufacturer' column
#start_index = df['Manufacturer'].isna().idxmax() if df['Manufacturer'].isna().any() else 0
start_index=0
max_attempts = 3  # Set the maximum number of attempts per tail number

# Start the loop from the 'start_index'
for iteration, tail_num in enumerate(df['TailNum'][start_index:], start=start_index):
    attempt = 0

    if iteration % 200==0 and iteration!=0:
        print(f'Beginning iteration #{iteration}', end="\r")
        print(f'Sleepy Time')
        driver.quit()
        time.sleep(300)
        # Set up the path to Firefox and geckodriver
        driver_path = r'C:\Users\me\dependencies\geckodriver-v0.33.0-win32\geckodriver.exe'
        firefox_path = r'C:\Program Files\Mozilla Firefox\firefox.exe'

        # Firefox options
        firefox_options = Options()
        firefox_options.binary_location = firefox_path

        # set up Firefox service with geckodriver
        firefox_service = FirefoxService(executable_path=driver_path)
```

```
    driver = webdriver.Firefox(service=firefox_service, options=firefox_options)

    url = 'https://registry.faa.gov/aircraftinquiry/search/nnumberinquiry'
    input_box_id = 'input-error'

    while attempt < max_attempts:
        try:
            print(f'Beginning iteration #{iteration}', end="\r")
            driver.get(url)
            input_box = driver.find_element(By.ID, input_box_id)
            input_box.clear()
            input_box.send_keys(tail_num)
            input_box.send_keys(Keys.RETURN)
            time.sleep(5)  # page load

            # get the html data
            soup = BeautifulSoup(driver.page_source, 'html.parser')

            # find the table with data
            table = soup.find('table', class_='devkit-table')

            if table is not None:
                # temp dictionary to hold data grabbed
                plane_data = {}

                for i, x in enumerate(table.find_all()):
                    if len(x) > 1:
                        for k in x:
                            if len(k) >= 4:
                                k = k.find_all()
                                if len(k) >= 4:
                                    plane_data[k[0].get_text(strip=True)] = k[1].get_text(strip=True)
                                    plane_data[k[2].get_text(strip=True)] = k[3].get_text(strip=True)

                # add values to df, using fallback keys if necessary
                df.loc[iteration, 'Manufacturer'] = plane_data.get('Manufacturer Name', 'Unknown')
                df.loc[iteration, 'Model'] = plane_data.get('Model', 'Unknown')
                df.loc[iteration, 'Expiration'] = plane_data.get('Expiration Date', 'missing')
                df.loc[iteration, 'Type Engine'] = plane_data.get('Type Engine', plane_data.get('Engine Model',
                df.loc[iteration, 'Type Aircraft'] = plane_data.get('Type Aircraft', 'Unknown')
                df.loc[iteration, 'Issue Date'] = plane_data.get('Certificate Issue Date', 'Unknown')
```

```
                df.loc[iteration, 'MFR Year'] = plane_data.get('MFR Year', plane_data.get('A/W Date', plane_data.

            break  # if successful, exit the while loop

        except WebDriverException as e:
            print(f"Error loading page on iteration #{iteration}: {e}")
            attempt += 1
            time.sleep(10)

        if attempt == max_attempts:
            print(f"Failed to load page after {max_attempts} attempts.")
            break  # exit the while loop
else:
    while attempt < max_attempts:
        try:
            print(f'Beginning iteration #{iteration}', end="\r")
            driver.get(url)
            input_box = driver.find_element(By.ID, input_box_id)
            input_box.clear()
            input_box.send_keys(tail_num)
            input_box.send_keys(Keys.RETURN)
            time.sleep(5)  # wait for page load

            # get html data
            soup = BeautifulSoup(driver.page_source, 'html.parser')

            # find table with data
            table = soup.find('table', class_='devkit-table')

            if table is not None:
                # temp dictionary to hold data grabbed
                plane_data = {}

                for i, x in enumerate(table.find_all()):
                    if len(x) > 1:
                        for k in x:
                            if len(k) >= 4:
                                k = k.find_all()
                                if len(k) >= 4:
                                    plane_data[k[0].get_text(strip=True)] = k[1].get_text(strip=True)
                                    plane_data[k[2].get_text(strip=True)] = k[3].get_text(strip=True)
```

```
                    # add values to df, using fallback keys if necessary
                    df.loc[iteration, 'Manufacturer'] = plane_data.get('Manufacturer Name', 'Unknown')
                    df.loc[iteration, 'Model'] = plane_data.get('Model', 'Unknown')
                    df.loc[iteration, 'Expiration'] = plane_data.get('Expiration Date', 'missing')
                    df.loc[iteration, 'Type Engine'] = plane_data.get('Type Engine', plane_data.get('Engine Model',
                    df.loc[iteration, 'Type Aircraft'] = plane_data.get('Type Aircraft', 'Unknown')
                    df.loc[iteration, 'Issue Date'] = plane_data.get('Certificate Issue Date', 'Unknown')
                    df.loc[iteration, 'MFR Year'] = plane_data.get('MFR Year', plane_data.get('A/W Date', plane_data.

                break  # if successful, exit the while loop

            except WebDriverException as e:
                print(f"Error loading page on iteration #{iteration}: {e}")
                attempt += 1
                time.sleep(900)

            if attempt == max_attempts:
                print(f"Failed to load page after {max_attempts} attempts.")
                break  # exit the while loop

# close driver after all iterations
driver.quit()
```

In [ ]:
```
df.drop(['Unnamed: 0.6', 'Unnamed: 0.4', 'Unnamed: 0.3', 'Unnamed: 0.2',
       'Unnamed: 0.1', 'Unnamed: 0','Unnamed: 0.5'],axis=1,inplace=True)
```

In [ ]:
```
df.to_csv('final_data.csv',index=False)
df.to_csv('final_data2.csv',index=False)
```

In [ ]:
```
set(df['Manufacturer'])
```

In [ ]:
```
set(df['Model'])
```

In [ ]:
```
df.columns
```

In [ ]:
```
set(df['Type Aircraft'])
```

In [ ]:
```
set(df['Type Engine'])
```

In [ ]:
```
df=pd.read_csv('final_data.csv')
```

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

In [ ]:
```
model_map = {
    # Cessna Models
    '150': 'Cessna 150',
    '150H': 'Cessna 150',
    '172M': 'Cessna 172',
    '172N': 'Cessna 172',
    '172S': 'Cessna 172',
    '182A': 'Cessna 182',
    '182P': 'Cessna 182',
    '182S': 'Cessna 182',
    '182T': 'Cessna 182',
    '206B': 'Cessna 206',
    '208B': 'Cessna 208',
    '25B': 'Cessna 25B',
    '310R': 'Cessna 310',
    '421C': 'Cessna 421',
    'U206G': 'Cessna 206',
    'T337G': 'Cessna 337',
    'PA-28-180': 'Piper PA-28',
    'PA-28-181': 'Piper PA-28',
    'PA-28R-200': 'Piper PA-28R',
    'PA-31T1': 'Piper PA-31T',
    # Boeing Models
    '737-301': 'Boeing 737',
    '737-317': 'Boeing 737',
    '737-322': 'Boeing 737',
    '737-3A4': 'Boeing 737',
    '737-3B7': 'Boeing 737',
    '737-3G7': 'Boeing 737',
    '737-3H4': 'Boeing 737',
    '737-3K2': 'Boeing 737',
    '737-3L9': 'Boeing 737',
    '737-3Q8': 'Boeing 737',
    '737-3T5': 'Boeing 737',
    '737-3Y0': 'Boeing 737',
    '737-401': 'Boeing 737',
    '737-4B7': 'Boeing 737',
```

```
    '737-522': 'Boeing 737',
    '737-5H4': 'Boeing 737',
    '737-705': 'Boeing 737',
    '737-76Q': 'Boeing 737',
    '737-790': 'Boeing 737',
    '737-7AD': 'Boeing 737',
    '737-7H4': 'Boeing 737',
    '737-7Q8': 'Boeing 737',
    '737-8': 'Boeing 737',
    '737-832': 'Boeing 737',
    '737-890': 'Boeing 737',
    '737-8FH': 'Boeing 737',
    '737-990': 'Boeing 737',
    '757-222': 'Boeing 757',
    '757-223': 'Boeing 757',
    '757-225': 'Boeing 757',
    '757-231': 'Boeing 757',
    '757-232': 'Boeing 757',
    '757-28A': 'Boeing 757',
    '757-2B7': 'Boeing 757',
    '757-2S7': 'Boeing 757',
    '767-201': 'Boeing 767',
    '767-300': 'Boeing 767',
    '767-322': 'Boeing 767',
    '767-323': 'Boeing 767',
    '767-332': 'Boeing 767',
    '767-3W0': 'Boeing 767',
    '767-432ER': 'Boeing 767',
    '777-222': 'Boeing 777',
    '777-232': 'Boeing 777',
    # Airbus Models
    'A318-111': 'Airbus A318',
    'A319-111': 'Airbus A319',
    'A319-112': 'Airbus A319',
    'A319-131': 'Airbus A319',
    'A319-132': 'Airbus A319',
    'A320-214': 'Airbus A320',
    'A320-231': 'Airbus A320',
    'A320-232': 'Airbus A320',
    'A321-211': 'Airbus A321',
    'A321-231': 'Airbus A321',
    'A321-253NX': 'Airbus A321',
```

```
    'A340-300': 'Airbus A340',
    # Bell Models
    '407': 'Bell 407',
    '414A': 'Bell 414',
    '429': 'Bell 429',
    # Others
    'A188B': 'Cessna A188B',
    'AEROSONDE MK 4.7G': 'Aerosonde 4.7G',
    'AGBOT': 'AGBOT',
    'AGRAS T30': 'DJI Agras T30',
    'AIR 2S': 'DJI Air 2S',
    'ANDREWS': 'Andrews',
    'ARAVA 101B': 'IAI Arava 101B',
    'AS 350B3': 'Airbus H125',
    'AS350B3': 'Airbus H125',
    'AT-402B': 'Air Tractor 402B',
    'AT-802A': 'Air Tractor 802A',
    'AVID FLYER MK IV': 'Avid Flyer Mk IV',
    'B200GT': 'Beechcraft King Air B200GT',
    'B300': 'Beechcraft King Air B300',
    'BB70Z': 'BB70Z',
    'BD-100-1A10': 'Bombardier Challenger 300',
    'CHALLENGER II': 'Quad City Challenger II',
    'CL-600-2B19': 'Bombardier CRJ200',
    'CL-600-2C10': 'Bombardier CRJ700',
    'CL-600-2D24': 'Bombardier CRJ900',
    'COPTER SONDE 2': 'Copter Sonde 2',
    'DA 42 NG': 'Diamond DA42',
    'DC-7BF': 'Douglas DC-7',
    'DC-9-82(MD-82)': 'McDonnell Douglas MD-82',
    'DC-9-83(MD-83)': 'McDonnell Douglas MD-83',
    'DHC-8-402': 'Bombardier Q400',
    'EMB-120': 'Embraer EMB 120',
    'EMB-120ER': 'Embraer EMB 120',
    'EMB-135KL': 'Embraer ERJ-135',
    'EMB-135LR': 'Embraer ERJ-135',
    'EMB-145': 'Embraer ERJ-145',
    'EMB-145LR': 'Embraer ERJ-145',
    'EMB-500': 'Embraer Phenom 100',
    'ERJ 190-100 IGW': 'Embraer E-Jet E190',
    'F85P-1': 'F85P-1',
    'FALCON XP': 'Falcon XP',
```

```
    'FALCON-XP': 'Falcon XP',
    'FAN JET FALCON': 'Dassault Falcon',
    'FAN JET FALCON SER D': 'Dassault Falcon',
    'G-1159A': 'Gulfstream III',
    'HS 125-700A': 'Hawker 700',
    'INSPIRE 1 PRO': 'DJI Inspire 1 Pro',
    'INSPIRE 2': 'DJI Inspire 2',
    'K35': 'Beechcraft Bonanza K35',
    'LA-4': 'Lake LA-4',
    'LC41-550FG': 'Columbia LC41-550FG',
    'MAVIC 2 PRO': 'DJI Mavic 2 Pro',
    'MBB-BK 117 C-2': 'Eurocopter EC145',
    'MD-88': 'McDonnell Douglas MD-88',
    'MODIFIED SONERAILL': 'Modified Sonerail',
    'MU-2B-35': 'Mitsubishi MU-2',
    'MU-300': 'Mitsubishi Diamond',
    'PC-12/45': 'Pilatus PC-12',
    'PHANTOM 3 ADVANCED': 'DJI Phantom 3 Advanced',
    'PHANTOM 4': 'DJI Phantom 4',
    'R44 II': 'Robinson R44',
    'R66': 'Robinson R66',
    'REV X': 'Rev X',
    'RV-8': 'Vans RV-8',
    'RV7': 'Vans RV-7',
    'S-50A': 'Sikorsky S-50A',
    'S-61L': 'Sikorsky S-61',
    'S2R-H80': 'Thrush S2R-H80',
    'SAAB 340B': 'Saab 340B',
    'SEAHUNTER': 'SeaHunter',
    'SENTRY HP': 'Sentry HP',
    'SF50': 'Cirrus Vision SF50',
    'SLING LSA': 'Sling LSA',
    'SPORTCRUISER': 'SportCruiser',
    'SR20': 'Cirrus SR20',
    'SR22T': 'Cirrus SR22',
    'STARDUSTER TOO': 'Starduster Too',
    'T-30': 'T-30',
    'TBM 700': 'Daher TBM 700',
    'UH-60A': 'Sikorsky UH-60 Black Hawk',
    'Unknown': 'Unknown',
    'V-BAT': 'V-BAT',
    'V2.2 WNTAC': 'V2.2 WNTAC',
```

```
    'V35-TC': 'Beechcraft Bonanza V35-TC',
    'VANS AIRCRAFT RV6': 'Vans RV-6',
    'VANS RV 8': 'Vans RV-8',
    'VANS RV-10': 'Vans RV-10',
    'X6A': 'X6A',
    'nan': 'Unknown' }
```

In [ ]:

```
additional_model_map = {
    # Additional Models
    '45': 'Learjet 45',
    '501': 'Cessna Citation I',
    '525': 'Cessna CitationJet',
    '525B': 'Cessna CitationJet',
    '550': 'Cessna Citation II',
    '65-A90': 'King Air A90',
    'A65': 'Beechcraft Bonanza A65',
    'BB70Z': 'BB70Z',  # Specific Model
    'COPTER SONDE 2': 'Copter Sonde 2',  # Specific Model
    'DA 42 NG': 'Diamond DA42 NG',
    'DC-7BF': 'Douglas DC-7',
    'DC-9-82(MD-82)': 'McDonnell Douglas MD-82',
    'DC-9-83(MD-83)': 'McDonnell Douglas MD-83',
    'DHC-8-402': 'Bombardier Dash 8 Q400',
    'F85P-1': 'F85P-1',  # Specific Model
    'G-1159A': 'Gulfstream III',
    'GV-SP (G550)': 'Gulfstream G550',
    'GVII-G600': 'Gulfstream G600',
    'HS 125-700A': 'Hawker 700',
    'LC41-550FG': 'Columbia 400',
    'MAVIC 2 PRO': 'DJI Mavic 2 Pro',
    'MBB-BK 117 C-2': 'Eurocopter EC145',
    'MODIFIED SONERAILL': 'Modified Sonerai II',
    'MU-2B-35': 'Mitsubishi MU-2',
    'MU-300': 'Mitsubishi Diamond',
    'PC-12/45': 'Pilatus PC-12',
    'PHANTOM 3 ADVANCED': 'DJI Phantom 3 Advanced',
    'PHANTOM 4': 'DJI Phantom 4',
    'REV X': 'Revolution Helicopter Mini-500',
    'RV7': 'Vans RV-7',
    'S-50A': 'Sikorsky S-50A',
    'S-61L': 'Sikorsky S-61',
```

```
    'S2R-H80': 'Thrush S2R-H80',
    'SEAHUNTER': 'SeaHunter',
    'SENTRY HP': 'Sentry HP',
    'SLING LSA': 'Sling LSA',
    'SPORTCRUISER': 'CZAW SportCruiser',
    'STARDUSTER TOO': 'Starduster Too',
    'T-30': 'T-30',  # Specific Model
    'UH-60A': 'Sikorsky UH-60 Black Hawk',
    'V-BAT': 'V-BAT',  # Specific Model
    'V2.2 WNTAC': 'V2.2 WNTAC',  # Specific Model
    'V35-TC': 'Beechcraft Bonanza V35-TC',
    'VANS AIRCRAFT RV6': 'Vans RV-6',
    'VANS RV 8': 'Vans RV-8',
    'VANS RV-10': 'Vans RV-10',
    'X6A': 'X6A',  # Specific Model
    'nan': 'Unknown',
    'Unknown': 'Unknown'
}

# merge original model map with the additional model map
full_model_map = {**model_map, **additional_model_map}
```

In [ ]:

```
manufacturer_map = {
    'AAI CORP DBA TEXTRON SYSTS UNM': 'TEXTRON',
    'AERIAL TECHNOLOGY INTL LLC': 'AERIAL TECHNOLOGY',
    'AIR TRACTOR INC': 'AIR TRACTOR',
    'AIRBUS': 'AIRBUS',
    'AIRBUS HELICOPTERS': 'AIRBUS',
    'AIRBUS HELICOPTERS INC': 'AIRBUS',
    'AIRBUS INDUSTRIE': 'AIRBUS',
    'ALASKA CTR FOR UNMANNED ACFT': 'ALASKA CTR UNMANNED',
    'AMERICAN AIRCRAFT INC': 'AMERICAN AIRCRAFT',
    'AMERICAN EUROCOPTER CORP': 'AMERICAN EUROCOPTER',
    'ANDREWS ALFRED': 'ANDREWS ALFRED',
    'BALONY KUBICEK SPOL SRO': 'BALONY KUBICEK',
    'BAUMAN RANDY': 'BAUMAN RANDY',
    'BEECH': 'BEECH',
    'BELL': 'BELL TEXTRON',
    'BELL HELICOPTER TEXTRON CANADA': 'BELL TEXTRON',
    'BELL TEXTRON CANADA LTD': 'BELL TEXTRON',
    'BENJAMIN D EGGERS': 'BENJAMIN EGGERS',
    'BOEING': 'BOEING',
```

```
    'BOMBARDIER INC': 'BOMBARDIER',
    'BRITISH AEROSPACE': 'BRITISH AEROSPACE',
    'CANADAIR': 'CANADAIR',
    'CESSNA': 'CESSNA',
    'CIRRUS DESIGN CORP': 'CIRRUS DESIGN',
    'CLANTON CHARLES M': 'CLANTON CHARLES',
    'COLUMBIA AIRCRAFT MFG': 'COLUMBIA AIRCRAFT',
    'CZECH SPORT AIRCRAFT A S': 'CZECH SPORT AIRCRAFT',
    'DASSAULT/SUD': 'DASSAULT',
    'DAVID A HOCK': 'DAVID HOCK',
    'DIAMOND AIRCRAFT IND GMBH': 'DIAMOND AIRCRAFT',
    'DJI': 'DJI',
    'DOUGLAS': 'DOUGLAS',
    'DRS TECHNOLOGIES': 'DRS TECHNOLOGIES',
    'E&E AIRCRAFT INC': 'E&E AIRCRAFT',
    'EMBRAER': 'EMBRAER',
    'EMBRAER-EMPRESA BRASILEIRA DE': 'EMBRAER',
    'FRIEDEMANN JON': 'FRIEDEMANN JON',
    'FRIEDER KEMMANN': 'FRIEDER KEMMANN',
    'GATES LEAR JET': 'GATES LEARJET',
    'GULFSTREAM AEROSPACE': 'GULFSTREAM',
    'GULFSTREAM AEROSPACE CORP': 'GULFSTREAM',
    'GULFSTREAM AMERICAN CORP.': 'GULFSTREAM',
    'HARRIS ROLAND W': 'HARRIS ROLAND',
    'ISRAEL AIRCRAFT INDUSTRIES': 'ISRAEL AIRCRAFT',
    'JASON CARVER': 'JASON CARVER',
    'JAY ANDING': 'JAY ANDING',
    'JEFFREY J BALLES SR': 'JEFFREY BALLES',
    'JERRY ST. ANDRE': 'JERRY ST. ANDRE',
    'JOHN BURGY': 'JOHN BURGY',
    'JOHNSON ROBERT M': 'JOHNSON ROBERT',
    'KILDALL GARY': 'KILDALL GARY',
    'KRANIG GLENN A': 'KRANIG GLENN',
    'LAKE': 'LAKE',
    'LAMBERT RICHARD': 'LAMBERT RICHARD',
    'LARRY A FRENCH': 'LARRY FRENCH',
    'LEARJET INC': 'LEARJET',
    'LEBLANC GLENN T': 'LEBLANC GLENN',
    'MCDONNELL DOUGLAS': 'MCDONNELL DOUGLAS',
    'MCDONNELL DOUGLAS AIRCRAFT CO': 'MCDONNELL DOUGLAS',
    'MITSUBISHI': 'MITSUBISHI',
    'MORSE GEORGE JR': 'MORSE GEORGE',
```

```
    'MOTT JEFFREY C': 'MOTT JEFFREY',
    'PILATUS AIRCRAFT LTD': 'PILATUS AIRCRAFT',
    'PIPER': 'PIPER',
    'RAVEN': 'RAVEN',
    'ROBINSON HELICOPTER CO': 'ROBINSON HELICOPTER',
    'ROBINSON HELICOPTER COMPANY': 'ROBINSON HELICOPTER',
    'SAAB-SCANIA': 'SAAB',
    'SEELBACK JAN/HERB': 'SEELBACK',
    'SHIELD AI INC': 'SHIELD AI',
    'SIKORSKY': 'SIKORSKY',
    'SKYWAYS AIR TRANSPORTATION INC': 'SKYWAYS AIR TRANSPORTATION',
    'SLING AIRCRAFT (PTY) LTD': 'SLING AIRCRAFT',
    'SOCATA': 'SOCATA',
    'STINNETT RON': 'STINNETT RON',
    'TEXTRON AVIATION INC': 'TEXTRON',
    'THRUSH AIRCRAFT INC': 'THRUSH AIRCRAFT',
    'TOM HALL': 'TOM HALL',
    'UNIVERSITY OF OKLAHOMA': 'UNIVERSITY OF OKLAHOMA',
    'Unknown': 'Unknown',
    'WOBIG WAYNE R': 'WOBIG WAYNE',
    'nan': 'Unknown'
}
```

In [ ]:
```
df['Manufac'] = df['Manufacturer'].replace(manufacturer_map)
df['Model_Group']=df['Model'].replace(full_model_map)
```

In [ ]:
```
columns_to_check = ['Expiration', 'Type Engine', 'Type Aircraft', 'Issue Date', 'MFR Year','Manufacturer','Model']

for x in columns_to_check:
    df[x]=df[x].fillna('missing')
```

In [ ]:
```
# list of columns to check for the specified values
columns_to_check = ['Expiration', 'Type Engine', 'Type Aircraft', 'Issue Date', 'MFR Year','Manufacturer','Model']

#define the values to consider as 'missing'
missing_values = ['', 'missing', 'Unknown', 'None',None,"NaN"]

#use `isin` to check for the missing values and `any(axis=1)` to check across the correct column
mask = df[columns_to_check].isin(missing_values).any(axis=1)

#invert mask to filter out the rows with missing values
```

```
df_cleaned = df[~mask]

#df_cleaned contains only the rows where none of the specified columns have missing or specified invalid values
len(df_cleaned)
```

In [ ]:
```
df=df_cleaned
```

In [ ]:
```
df['CertYear']=df['Issue Date'].str[-4:].astype(int)
```

In [ ]:
```
df['ExpYear']=df['Expiration'].str[-4:].astype(int)
```

In [ ]:
```
df.drop(['Model','Expiration','Issue Date','Manufacturer','CertYear','ExpYear'],axis=1,inplace=True)
```

In [ ]:
```
df.to_csv('CompletedDataSet1.csv')
df.to_csv('CompletedDataSet2.csv')
```