Homework 1 Written Answers

Data Engineering 300
Aden Benson
4/18/2025

Task 1

Prior to any imputation/transformation, I found the number of rows that we would keep if we were not able to clean any of the columns given in the assignment. Without cleaning, there would be 132037 rows left in our dataset.

```
[3]: # First, check how many rows we have left if we were to delete all rows with missing data at the start.
removed_data=data.dropna(subset=['CARRIER','CARRIER_NAME','MANUFACTURE_YEAR','NUMBER_OF_SEATS','CAPACITY_IN_POUNDS','AIRLINE_ID',
                                 'MANUFACTURER','MODEL','AIRCRAFT_STATUS','OPERATING_STATUS'])

print('Rows left after removing missing values with no cleaning:', removed_data.shape[0])

Rows left after removing missing values with no cleaning: 132037
```

Handling CARRIER

The CARRIER column had missing values in the form of NaN. After looking at the clue given in the assignment document, I found that all rows where CARRIER was NaN also had a CARRIER_NAME value of North American Airlines. The acronym for this company is actually NaN, which tells us that this is meant to be a string. However, Python interprets this as a missing value, so we can set the entire column to be strings and fix this issue. Below, we see all missing values in the CARRIER column have CARRIER_NAME set to North American Airlines.

```
[8]: data[pd.isna(data['CARRIER'])]['CARRIER_NAME'].unique()

[8]: array(['North American Airlines'], dtype=object)
```

Handling CARRIER_NAME

All missing values in the CARRIER_NAME column come when CARRIER is either “OH” or “L4”, as shown below.

```
[12]: data[pd.isna(data['CARRIER_NAME'])]['CARRIER'].value_counts()

[12]: CARRIER
      OH    97
      L4     8
      Name: count, dtype: int64
```

Based on this information, we can analyze all rows where CARRIER is either OH or L4, and try and impute based on these findings. First, when we analyze all entries with L4, we find that CARRIER_NAME is either missing or listed as “Lynx Aviation d/b/a Frontier Airlines”. Based on this, we can impute all missing values in the CARRIER_NAME column when CARRIER is L4 as this Lynx Aviation string.

[14]:

	YEAR	CARRIER	CARRIER_NAME	MANUFACTURE_YEAR	UNIQUE_CARRIER_NAME	S
11465	2007	L4	NaN	2007.0	NaN	
11466	2007	L4	NaN	2007.0	NaN	
11467	2007	L4	NaN	2007.0	NaN	
11468	2007	L4	NaN	2007.0	NaN	
11469	2007	L4	NaN	2007.0	NaN	
11470	2007	L4	NaN	2007.0	NaN	
11471	2007	L4	NaN	2007.0	NaN	
11472	2007	L4	NaN	2007.0	NaN	
18739	2008	L4	Lynx Aviation d/b/a Frontier Airlines	2007.0	Lynx Aviation d/b/a Frontier Airlines	
18740	2008	L4	Lynx Aviation d/b/a Frontier Airlines	2007.0	Lynx Aviation d/b/a Frontier Airlines	
18741	2008	L4	Lynx Aviation d/b/a Frontier Airlines	2007.0	Lynx Aviation d/b/a Frontier Airlines	

When we look at all rows where CARRIER_NAME is missing and CARRIER is listed as OH, it becomes more complicated. All rows with OH either have CARRIER_NAME as ‘Comair Inc.’, or ‘PSA Airlines Inc.’. However, after analyzing these rows, I found that the missing entries had similar serial number and tail number structures as the rows with CARRIER_NAME listed as Comair Inc. Thus, we can impute these values.

[17]:
```
# Next, examine columns with OH Carrier
data[data['CARRIER']=='OH']['CARRIER_NAME'].unique()
```

[17]:
```
array(['Comair Inc.', nan, 'PSA Airlines Inc.'], dtype=object)
```

Handling MANUFACTURE_YEAR

When we got to MANUFACTURE_YEAR, I actually found five instances of missing data. MANUFACTURE_YEAR had entries of NaN, 0, 1197, 2756, and 1900, which are all not valid entries for the manufacturing year of an airplane.

```
[21]: # First, check how missing information is shown.
      data['MANUFACTURE_YEAR'].unique()

[21]: array([2003., 2004., 2005., 1944., 1945., 1954., 1955., 1956., 1957.,
             1958., 1976., 1977., 1978., 1988., 1992., 1997., 1966., 1967.,
             1968., 1969., 1970., 1971., 1980., 1982., 1987., 1990., 1991.,
             1993., 1994., 1995., 1996., 1998., 1999., 2000., 2001., 2002.,
             2006., 1979., 1981., 1985., 1975., 1984., 1983., 1986., 1989.,
             1972., 1973., 1974., 1965., 1953., 1952., 1959., 2007., 2008.,
             2009., 1961., 2010.,    0., 2011., 2012., 1964., 2013., 2014.,
             2015., 1900., 2016., 1942., 2017., 2018., 2019., 2020., 2021.,
             1197., 2022.,   nan, 2756., 2023.])
```

First, I checked the years 1197, 2756, and 1900. All three of these instances were easily rectified, as I was able to look at the serial numbers for each of these rows and find the correct manufacturing year listed on the other entries for that serial number.

When analyzing entries with MANUFACTURE_YEAR listed as NaN, there were three entries with missing values.

```
[32]: # Check nan
      data[pd.isna(data['MANUFACTURE_YEAR'])]
```

[32]:

	YEAR	CARRIER	CARRIER_NAME	MANUFACTURE_YEAR	UNIQUE_CARRIER_NAME	SERIAL_NUMBER
116651	2022	5Y	Atlas Air Inc.	NaN	Atlas Air Inc.	26259
124680	2023	9E	Endeavor Air Inc.	NaN	Endeavor Air Inc.	10134
124681	2023	9E	Endeavor Air Inc.	NaN	Endeavor Air Inc.	10182

I was able to rectify the two Endeavor Air entries using the serial number referencing technique, as they were both manufactured in 2004. However, the Atlas Air entry had no other rows with that serial number, and required a method of imputation learned in class. To do this, I found the median manufacturing year of all instances where MODEL was listed as “B747-400”. I then imputed this value for our missing instance.

Finally, it was time to analyze entries with 0 listed for MANUFACTURE_YEAR. This was a longer list, where I first used the serial number referencing strategy. There were rows left over that still contained a missing entry, so I used the same strategy as the Atlas Air entry for these rows, finding the median manufacturing year for their model and imputing this value.

Handling NUMBER_OF_SEATS

For the column NUMBER_OF_SEATS, there were two entries that I thought could refer to missing values: NaN and 0. However, after analyzing all entries with NUMBER_OF_SEATS listed as 0, I found that the MODEL listed these as cargo planes, so this made sense. Thus, I only decided to impute values listed as NaN. This dataset also consisted of only cargo planes, so it was easy to just impute 0 for these missing values as well. The figure below shows how each of these model names contains the word "Cargo".

```
[43]: data[pd.isna(data['NUMBER_OF_SEATS'])]
```

[43]:

RAFT_STATUS	OPERATING_STATUS	NUMBER_OF_SEATS	MANUFACTURER	AIRCRAFT_TYPE	MODEL	C
b	Y	NaN	BOEING	6252	767-232SFCARGO	
b	Y	NaN	BOEING	6262	767-338ERCARGO	
b	Y	NaN	BOEING	6262	767-338ERCARGO	
b	Y	NaN	BOEING	6262	767-338ERCARGO	
b	Y	NaN	BOEING	6262	767-323ERCARGO	
b	Y	NaN	BOEING	6262	767-323ERCARGO	
b	Y	NaN	BOEING	6262	767-323ERCARGO	

Handling CAPACITY_IN_POUNDS

All missing entries in the CAPACITY_IN_POUNDS column come from the same three Carriers, as shown below.

```
[47]: # First investigate missing rows
      data[pd.isna(data['CAPACITY_IN_POUNDS'])]['CARRIER_NAME'].value_counts()

[47]: CARRIER_NAME
      Federal Express Corporation    96
      Atlas Air Inc.                  3
      Spirit Air Lines                2
      Name: count, dtype: int64
```

We can also see that all missing entries in this column come from similar models.

```
[49]: print(data[(pd.isna(data['CAPACITY_IN_POUNDS']))]['MODEL'].value_counts())

      MODEL
      MD-11            57
      DC-10-10         26
      DC-10-30         13
      B767-300          2
      A-320-PSGRneo     2
      B767-200          1
      Name: count, dtype: int64
```

Thus, to fix these missing values I found the average number of seats for each nonzero entry associated with each of the above models, and then imputed this value into all NaN's. I felt using the mean imputation method was the best idea, as many of these entries are similar to each other and there is likely little variation between the rows with missing entries.

Handling AIRLINE_ID

I instantly found that all Carrier names with missing AIRLINE_ID values are the Carrier names from above. There is likely a connection between the two missing columns.

```
[53]: # All missing values in our airline_id column are Lynx Aviation or Comair,
      data[pd.isna(data['AIRLINE_ID'])]['CARRIER_NAME'].unique()

[53]: array(['Lynx Aviation d/b/a Frontier Airlines', 'Comair Inc.'],
            dtype=object)
```

First, I found that all Lynx Aviation entries in our dataset had AIRLINE_ID entries of 21217. Thus, we were able to simply impute this value into our missing entries with Lynx Aviation

listed as the Carrier.

```
[54]: # Below, we see all Lynx Aviation planes have an airline id of 21217. Update these when the value is missing
      data[data['CARRIER_NAME']=='Lynx Aviation d/b/a Frontier Airlines']
```

[54]:

AIRCRAFT_STATUS	OPERATING_STATUS	NUMBER_OF_SEATS	MANUFACTURER	AIRCRAFT_TYPE	MODEL	CAPACITY_IN_POUNDS	ACQUISITION_DATE	AIRLINE_ID
B	Y	74.0	DEHAVILLAND	NaN	DASH8-Q4	19000.0	7/19/2007 12:00:00 AM	NaN
O	Y	74.0	DEHAVILLAND	NaN	DASH8-Q4	19000.0	8/28/2007 12:00:00 AM	NaN
B	Y	74.0	DEHAVILLAND	NaN	DASH8-Q4	19000.0	10/12/2007 12:00:00 AM	NaN
B	Y	74.0	DEHAVILLAND	NaN	DASH8-Q4	19000.0	10/19/2007 12:00:00 AM	NaN
B	Y	74.0	DEHAVILLAND	NaN	DASH8-Q4	19000.0	10/25/2007 12:00:00 AM	NaN
B	Y	74.0	DEHAVILLAND	NaN	DASH8-Q4	19000.0	10/31/2007 12:00:00 AM	NaN
O	Y	74.0	DEHAVILLAND	NaN	DASH8-Q4	19000.0	12/31/2007 12:00:00 AM	NaN
B	Y	74.0	DEHAVILLAND	NaN	DASH8-Q4	19000.0	12/31/2007 12:00:00 AM	NaN
B	Y	74.0	DEHAVILLAND	NaN	DASH8-Q4	19000.0	7/19/2007 12:00:00 AM	21217.0
O	Y	74.0	DEHAVILLAND	NaN	DASH8-Q4	19000.0	8/28/2007 12:00:00 AM	21217.0
B	Y	74.0	DEHAVILLAND	NaN	DASH8-Q4	19000.0	10/12/2007 12:00:00 AM	21217.0

Next, I analyzed the missing entries with Comair Inc. listed as the Carrier name. I found that Comair Inc. had an airline id of 20417 before 2013, a missing value in 2013, and an airline id of 20397 after 2013. Thus, we can impute either of these values into the missing entries since we don't know when the change actually started. Based on this information, I imputed all missing entries with Comair Inc. listed as the Carrier name as 20417.

Task 2

Standardizing MANUFACTURER

To standardize the manufacturer column, I first analyzed the list of all unique entries, and found many common strings present. The list of strings we used to create labels for the MANUFACTURER column is shown below.

```
manufacturers=['Boeing', 'Embraer', 'Bombardier', 'Airbus', 'McD',
               'Cessna', 'Canadair', 'Douglas', 'Dehavilland','ATR',
               'Gulfstream','Saab','Convair','Lockheed','Dassault',
               'Dornier','Easyjet','Curtiss','Iberia','SMBC','GECAS',
               'Aercap','BAE','Piper','Hawker', 'Fokker','ALC','Peach','Raytheon',
               'ICBC','JPLease','Lear','IAI','Beechcraft']
```

I then searched through each row in the MANUFACTURER column, and found if a string was contained from our list above. If it was, I substituted that label for our standardized labels above. I also performed standardization tasks after this that weren't caught by our initial string matching technique, shown below.

```
[62]: # Handle other standardizations in the manufacture column that aren't caught by our string matching technique
for i in range(data.shape[0]):
    # Aerospatiale=ATR
    if 'aero' in data.loc[i, 'MANUFACTURER'].lower():
        data.loc[i, 'MANUFACTURER']='ATR'

    # Challenger is Bombardier
    if 'chall' in data.loc[i, 'MANUFACTURER'].lower():
        data.loc[i, 'MANUFACTURER']='Bombardier'

    # Dassault is listed as Dassalt sometimes
    if 'dassult' in data.loc[i, 'MANUFACTURER'].lower():
        data.loc[i, 'MANUFACTURER']='Dassault'

    # IAI is spelled out
    if 'israel' in data.loc[i, 'MANUFACTURER'].lower():
        data.loc[i, 'MANUFACTURER']='IAI'

    # Raytheon is misspelled
    if 'raetheon' in data.loc[i, 'MANUFACTURER'].lower():
        data.loc[i, 'MANUFACTURER']='Raytheon'

    # Boeing is shortened
    if 'b757' in data.loc[i, 'MANUFACTURER'].lower():
        data.loc[i, 'MANUFACTURER']='Boeing'
```

Standardizing MODEL

The MODEL column was very messy, so I used a similar string-matching technique as I did above. First, I performed a basic clean over all entries in this row, getting rid of extra spaces, making all letters lowercase, and replacing any underscores with dashes. I then created a mapping dictionary for each of the ten ten most prominent manufacturers, structured as shown below.

```
# Next, create a mapping dictionary for each of the top 10 manufacturers
boeing_mapping={
    '717': 'Boeing 717', '727': 'Boeing 727', '73': 'Boeing 737', '74': 'Boeing 747', '757': 'Boeing 757',
    '767': 'Boeing 767', '777': 'Boeing 777', '787': 'Boeing 787'
}
```

Using this dictionary, I looked through each entry from these top ten manufacturers, and found if the key was present in any model names. If it was, then I substituted that model name for the standardized label given in our dictionaries. This process brought our unique number of models down from 1340 to 341, decreasing the number of distinct labels by almost 1000.

```
data['MODEL'].value_counts()

MODEL
Boeing 737            29196
Airbus 320            18881
Boeing 757             9687
Boeing 767             6894
Embraer ERJ 145        6011
                      ...
raytheonbeechcraft        1
ce-560-560xls             1
c-206pass                 1
dc-9-33cargo              1
westwind                  1
Name: count, Length: 341, dtype: int64
```

Standardizing AIRCRAFT_STATUS

Initial of the AIRCRAFT_STATUS column shows that we have values of O, A, B, and L, some uppercase and others lowercase. O means owned, A means capital lease, B means operating lease, and L is an unknown value.

```
data['AIRCRAFT_STATUS'].value_counts()

AIRCRAFT_STATUS
O    79487
b    30852
B    12699
a     7804
A     1330
L      122
o       19
Name: count, dtype: int64
```

I began by standardizing all entries as uppercase. However, I then imputed all entries listed as L to NaN, since these are technically unknown values and we cannot use them.

Standardize OPERATING_STATUS

Initial analysis of OPERATING_STATUS shows that we only have values of Y and N, along with some lowercase and missing values.

```
data['OPERATING_STATUS'].value_counts()
```

```
OPERATING_STATUS
Y    126577
N      5664
y        71
          1
```

To fix this, I simply set all values as uppercase in this row, along with setting the missing value to NaN so we can remove it in the next step.

Task 3

After completing all imputation and standardization, I found that there were 132179 rows that did not have any missing values in the columns we analyzed.

```
# See how many rows are left if we delete missing values after cleaning
removed_data_post_cleaning=data.dropna(subset=['CARRIER','CARRIER_NAME','MANUFACTURE_YEAR','NUMBER_OF_SEATS','CAPACITY_IN_POUNDS','AIRLINE_ID',
                                               'MANUFACTURER','MODEL','AIRCRAFT_STATUS','OPERATING_STATUS'])

print('Rows left after removing missing values after cleaning:', removed_data_post_cleaning.shape[0])
```

```
Rows left after removing missing values after cleaning: 132179
```

This means that our efforts lead to 142 rows being saved from deletion due to missing values. This number also would have been much higher if I had chosen to leave the unknown entries of "L" in for the AIRCRAFT_STATUS column.

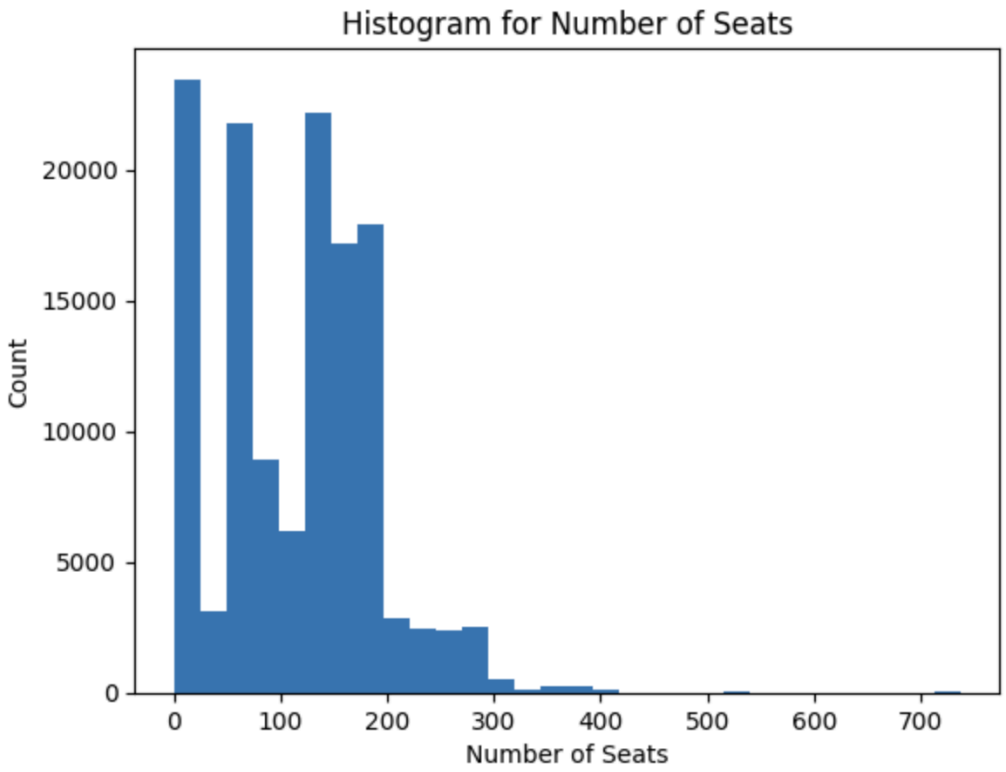
Task 4

First, we analyze the skewness of NUMBER_OF_SEATS and CAPACITY_IN_POUNDS, which is shown below. Since both of these values are positive, our data in these columns are skewed to the right, with CAPACITY_IN_POUNDS further skewed right. This is consistent with the information shown in the histograms below.

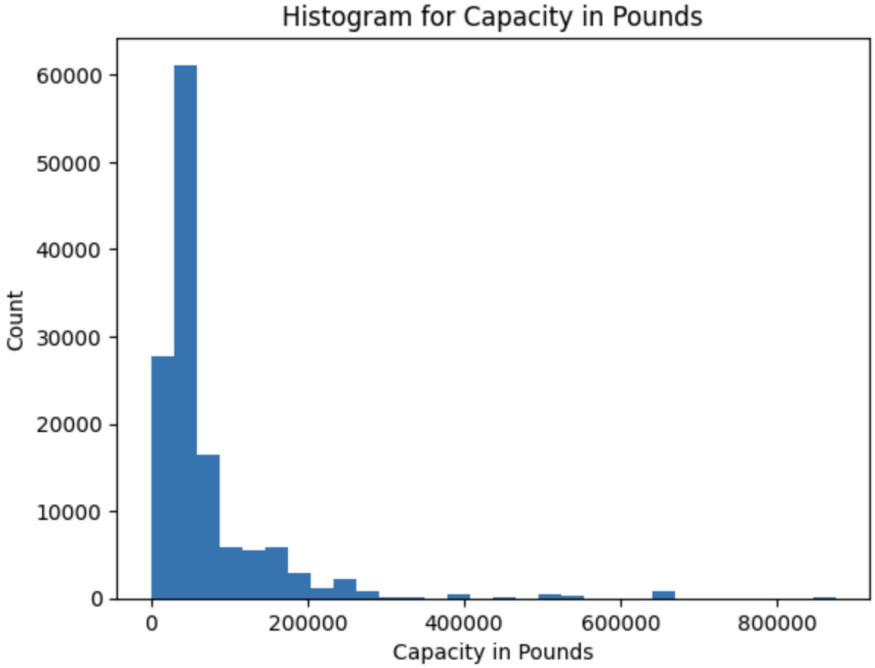
```
# Check skewness for number of seats and capacity in pounds
print('Skewness in number of seats:',data['NUMBER_OF_SEATS'].skew())
print('Skewness in capacity in pounds:',data['CAPACITY_IN_POUNDS'].skew())
```

```
Skewness in number of seats: 0.4316454677404661
Skewness in capacity in pounds: 4.067256628764957
```

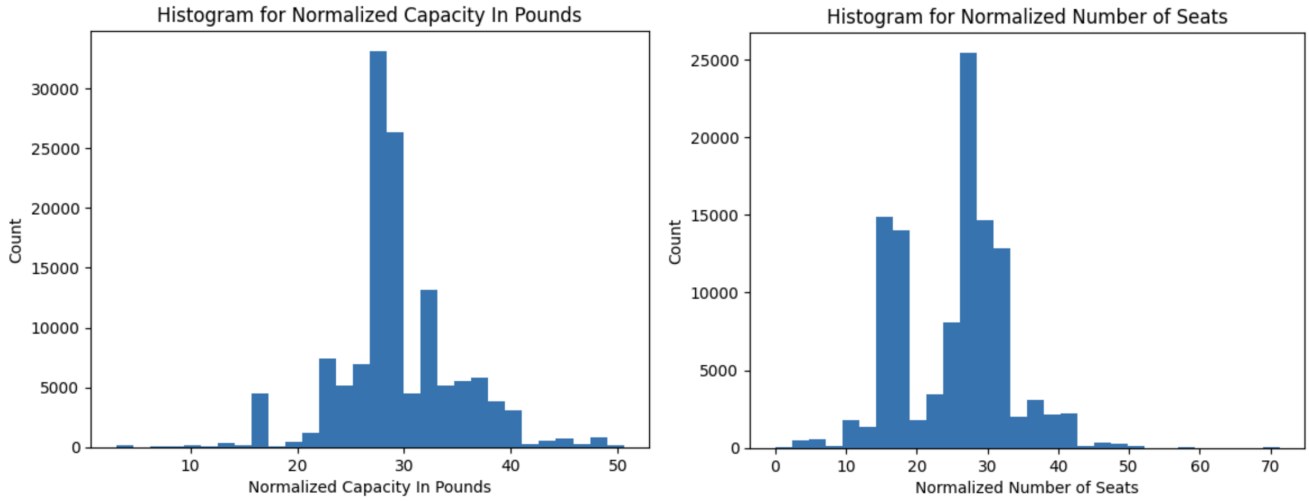
The histogram for NUMBER_OF_SEATS is shown below, where the majority of our data is on the left side, leading to the slightly right skew.



The histogram for CAPACITY_IN_POUNDS is shown in the next figure, and we see that it is even further skewed to the right than NUMBER_OF_SEATS. This is consistent with the constant values we found for the skew on each column.



After analyzing these rows, I used Box-Cox transformation to create 2 new normalized columns for NUMBER_OF_SEATS and CAPACITY_IN_POUNDS. The histograms for these new columns are shown below.



The two histograms above get rid of the skewness that was in our two initial columns, as they have both become much more symmetrical and similar to a normal distribution. The reason that we do this is to make the data more appropriate for modeling techniques like linear regression or clustering, which perform better on normally distributed inputs. This allows any user for our dataset to perform better analysis.

Task 5

The final task consists of feature engineering, where we create a qualitative column named SIZE that assigns labels to the quantitative column of NUMBER_OF_SEATS. After this is done, we can analyze the proportions of the columns OPERATING_STATUS and AIRCRAFT_STATUS for each size of airplane. These proportions are shown below.

```
SIZE    OPERATING_STATUS
LARGE   Y                 0.962423
        N                 0.037577
MEDIUM  Y                 0.936989
        N                 0.063011
SMALL   Y                 0.959893
        N                 0.040107
XLARGE  Y                 0.973358
        N                 0.026642
Name: proportion, dtype: float64
```

```
SIZE    AIRCRAFT_STATUS
LARGE   0                 0.642146
        B                 0.246293
        A                 0.111562
MEDIUM  B                 0.536721
        0                 0.405737
        A                 0.057543
SMALL   0                 0.720795
        B                 0.242823
        A                 0.036382
XLARGE  0                 0.695000
        B                 0.235389
        A                 0.069611
Name: proportion, dtype: float64
```

Based on these figures, we see that OPERATING_STATUS tends to stay the same regardless of the size of the airplane. All sizes of airplanes have an operating status of Y between 93%-97% of the time, showing a lack of influence between the two columns. However, AIRCRAFT_STATUS tends to differ based on the airplane size, as we see that the small and extra large airplanes are more likely to be owned than medium or large airplanes. Since medium and large airplanes have higher proportions of "A" and "B", we know that these tend to be leased more than their counterparts. The histograms below prove our conclusions in a visualization, as OPERATING_STATUS tends to stay the same while AIRCRAFT_STATUS is dependent on the size of the plane.