Exploratory Data Analysis

Team 01 DSCI-522

Group Project: Credit_Approval_Prediction

Data

Data used for this project comes from UC Irvine's Machine Learning Repository https://archive-beta.ics.uci.edu/dataset/27/credit+approval The dataset contains data on Japanese Credit Card screeing for credit card applications where all attribute names and values have been anonymized in order to protect the confidentiality of the applicants. Features contained in the dataset include continuous features and categorical features named A1-A16. The target feature is A16 containing values `+` or `-` indicating wheather the candidate was approved or not.

In [1]:
```
# Importing needed libraries
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
```

Read in data We are creating a dataframe with the data and looking at the top few rows in order to get an initial feel for what the data looks like.

In [4]:
```
df = pd.read_csv("crx.csv", encoding="utf-8")
```

With df.shape we can see that there are 690 observations of the 16 features in the dataset. So this is not a very big dataset.

In [3]:
```
df.shape
```

Out[3]:
```
(690, 16)
```

Splitting the data

Before exploring the data we create Train and Test splits with 20% of the data used as test data and 80% used as training data.

In [5]:
```
train_df, test_df = train_test_split(df, test_size=0.2, random_state=522)
```

Initial look at data The dataset has 16 columns with 522 values each all of which are non-null.

In [7]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 552 entries, 146 to 288
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   A1      552 non-null    object 
 1   A2      552 non-null    object 
 2   A3      552 non-null    float64
 3   A4      552 non-null    object 
 4   A5      552 non-null    object 
 5   A6      552 non-null    object 
 6   A7      552 non-null    object 
 7   A8      552 non-null    float64
 8   A9      552 non-null    object 
 9   A10     552 non-null    object 
 10  A11     552 non-null    int64  
 11  A12     552 non-null    object 
 12  A13     552 non-null    object 
 13  A14     552 non-null    object 
 14  A15     552 non-null    int64  
 15  A16     552 non-null    object 
dtypes: float64(2), int64(2), object(12)
memory usage: 73.3+ KB
```

Take a look at individual columns of the dataset

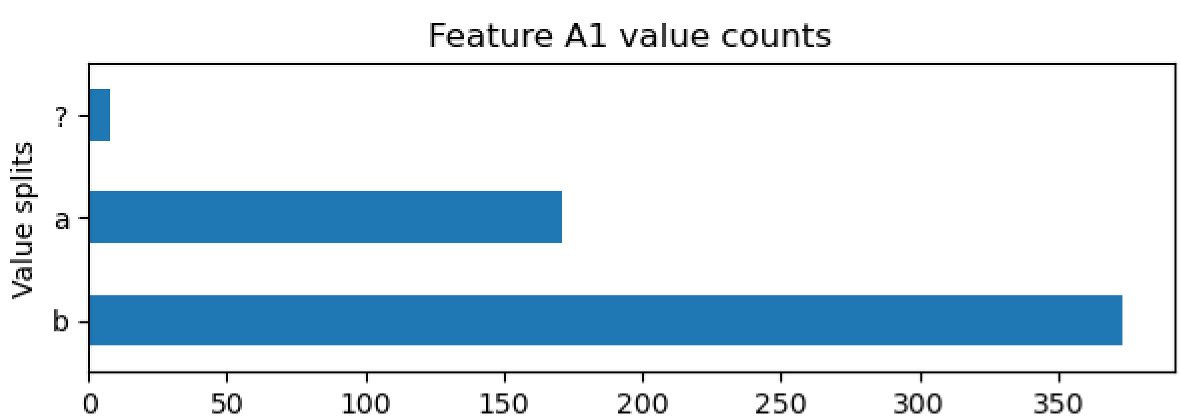
In order to better understand what the data contains we want to look at the individual columns separately. We'll first look at the values distribution for the categorical columns. After taking an initial look at feature `A1` becomes clear that missing values are replaced with a `?`. This will require us to preprocess the data in order to account for these missing values. Missing values have been replaced in some of the other categorical columns, e.g. `A4`, `A5`, `A6` and `A7`. These will need to be replaced with null values in preprocessing. The other categorical columns are clean of missing values.

Missing values exist also in some of the numerical columns like `A2` and `A14`, where the missing values have been replaced with a `?`. This change makes Pandas evaluate the two columns as categorical although they are numeric.

The target feature `A16` contains values of `+` and `-` for positive or negative approval decision on the credit card application. This column is well balanced with about 300 positive and 390 negative values, which makes the target well balanced.

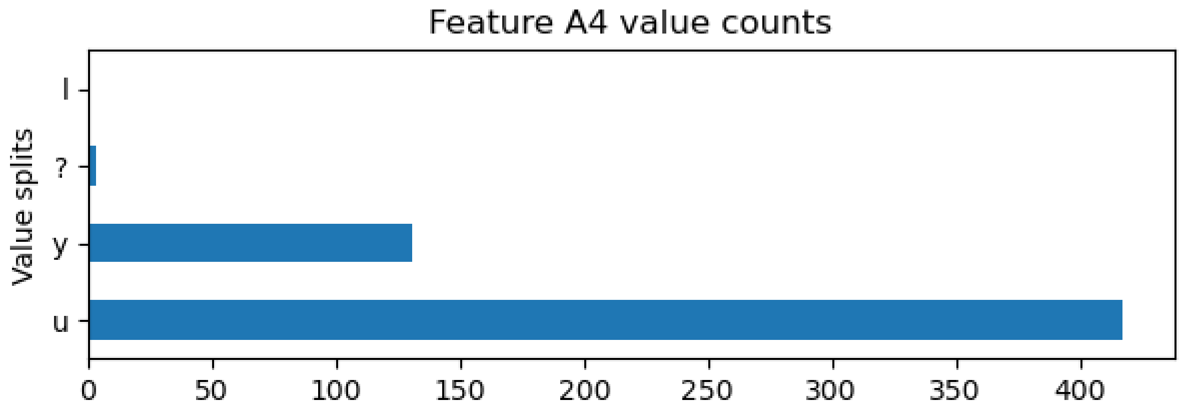
In [10]:
```
train_df['A1'].value_counts().plot(kind='barh', figsize=(7,2), title = "Feature A1 value counts".
```

Out[10]:
```
<AxesSubplot: title={'center': 'Feature A1 value counts'}, ylabel='Value splits'>
```



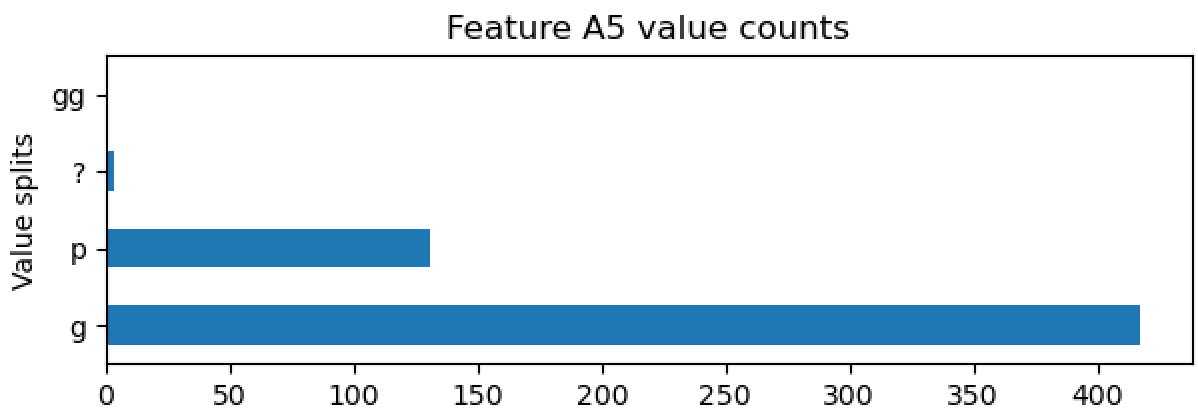
In [11]:
```
train_df['A4'].value_counts().plot(kind='barh', figsize=(7,2), title = "Feature A4 value counts"
```

Out[11]: <AxesSubplot: title={'center': 'Feature A4 value counts'}, ylabel='Value splits'>



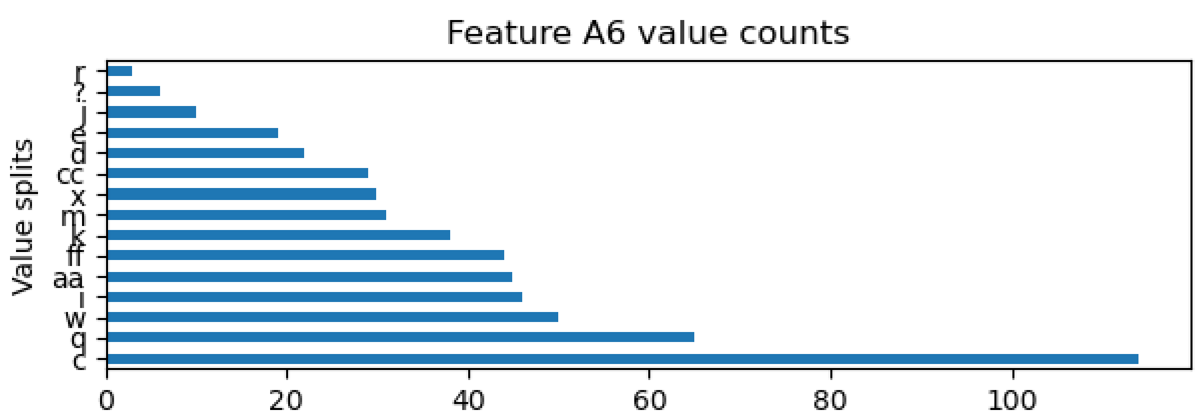
In [12]:
```
train_df['A5'].value_counts().plot(kind='barh', figsize=(7,2), title = "Feature A5 value counts"
```

Out[12]: <AxesSubplot: title={'center': 'Feature A5 value counts'}, ylabel='Value splits'>



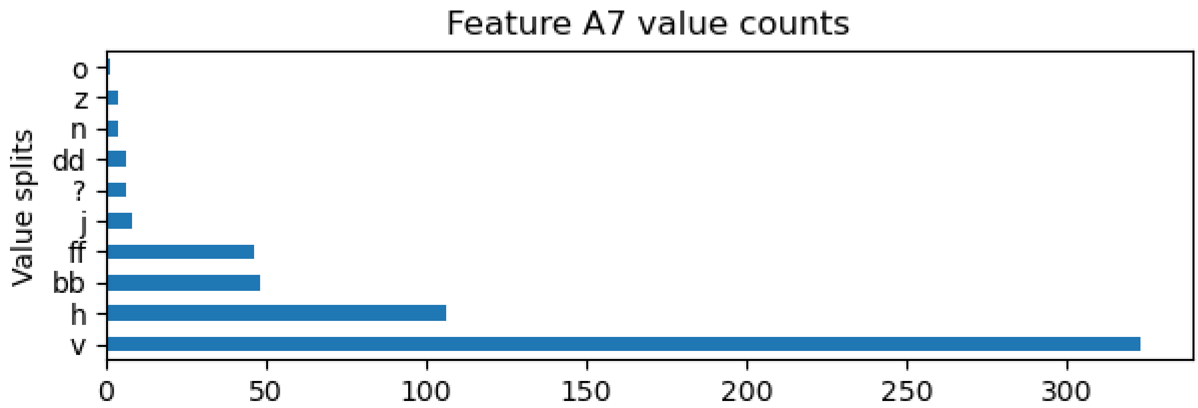
In [13]:
```
train_df['A6'].value_counts().plot(kind='barh', figsize=(7,2), title = "Feature A6 value counts"
```

Out[13]: <AxesSubplot: title={'center': 'Feature A6 value counts'}, ylabel='Value splits'>



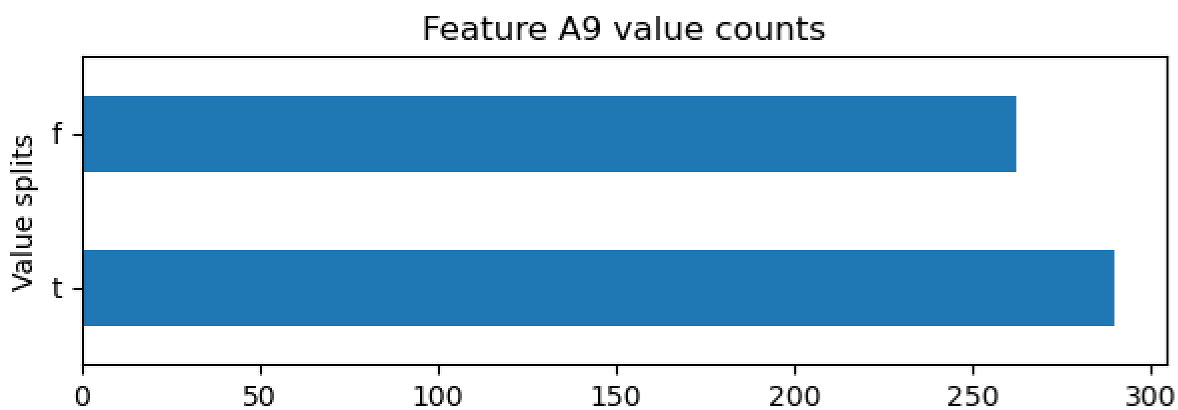
In [14]:
```
train_df['A7'].value_counts().plot(kind='barh', figsize=(7,2), title = "Feature A7 value counts"
```

Out[14]: <AxesSubplot: title={'center': 'Feature A7 value counts'}, ylabel='Value splits'>



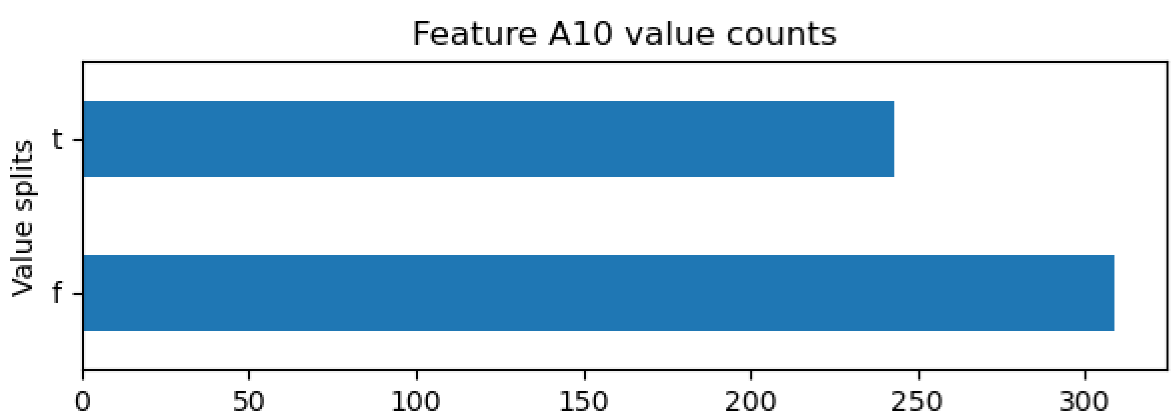

```
In [15]: train_df['A9'].value_counts().plot(kind='barh', figsize=(7,2), title = "Feature A9 value counts",
```

```
Out[15]: <AxesSubplot: title={'center': 'Feature A9 value counts'}, ylabel='Value splits'>
```



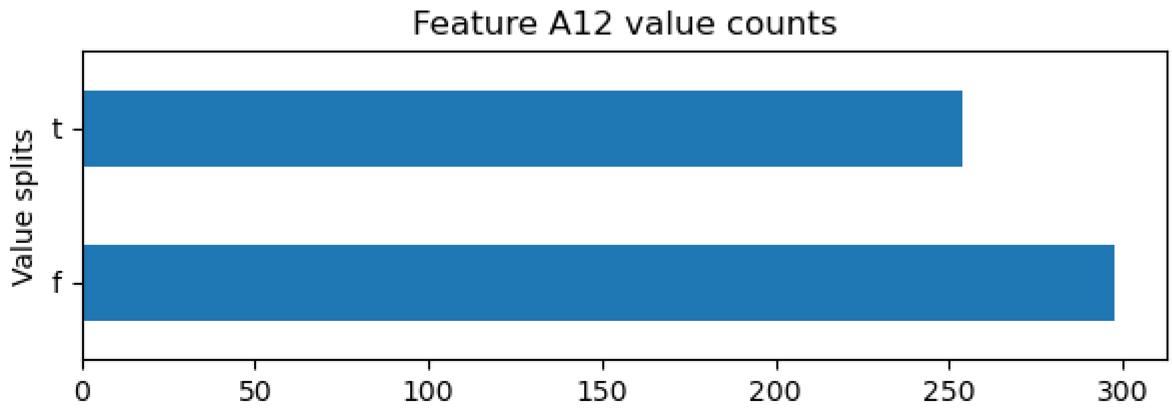

```
In [16]: train_df['A10'].value_counts().plot(kind='barh', figsize=(7,2), title = "Feature A10 value counts
```

```
Out[16]: <AxesSubplot: title={'center': 'Feature A10 value counts'}, ylabel='Value splits'>
```



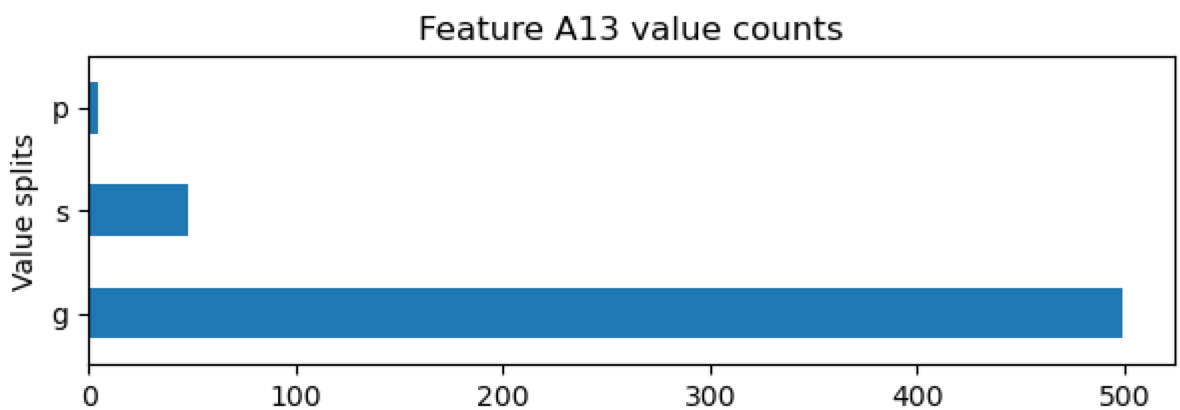

```
In [17]: train_df['A12'].value_counts().plot(kind='barh', figsize=(7,2), title = "Feature A12 value counts
```

```
Out[17]: <AxesSubplot: title={'center': 'Feature A12 value counts'}, ylabel='Value splits'>
```



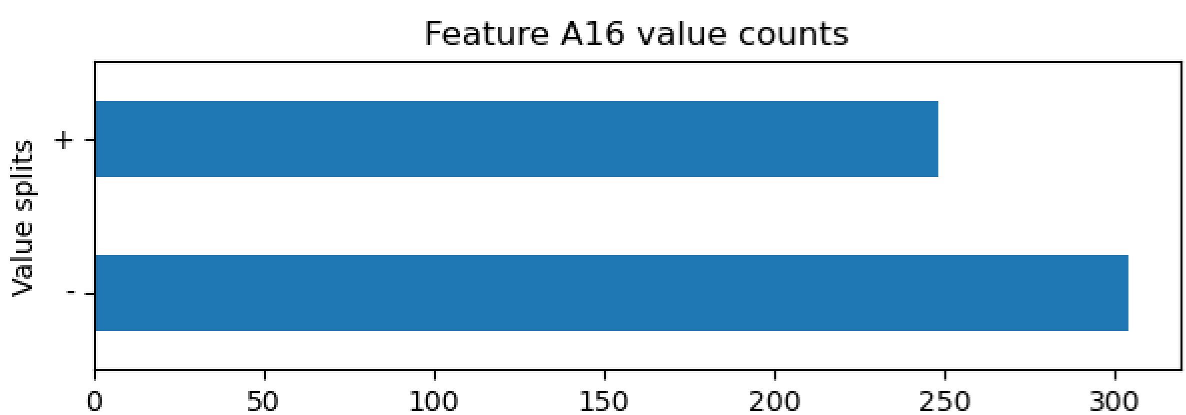
In [18]:
```
train_df['A13'].value_counts().plot(kind='barh', figsize=(7,2), title = "Feature A13 value count
```

Out[18]: `<AxesSubplot: title={'center': 'Feature A13 value counts'}, ylabel='Value splits'>`



In [19]:
```
train_df['A16'].value_counts().plot(kind='barh', figsize=(7,2), title = "Feature A16 value count
```

Out[19]: `<AxesSubplot: title={'center': 'Feature A16 value counts'}, ylabel='Value splits'>`



Replace `?` with `np.nan`

In order to gain better understanding of the missing values, we are replacing the question mark in place of missing values with `np.nan`. Now when we run an evaluation of the missing values, we get a better picture of the features containing `nan` values:

In [20]:
```
train_df = train_df.replace('?', np.nan)
```

In [21]:
```
train_df.isnull().sum()
```

Out[21]:
```
A1      8
A2     10
A3      0
A4      3
A5      3
A6      6
A7      6
A8      0
A9      0
A10     0
A11     0
A12     0
A13     0
A14     8
A15     0
A16     0
dtype: int64
```

Convert columns A2 and A14 to float

Originaly the two columns `A2` and `A14` were interpreted as categorical by Pandas due to the `?` in place of missing values. Now that we have replaced `?` with `np.nan` we can convert these columns as type `float` in order to better understand their values. We can see in the description of the numeric columns that the values have different degree of variance and min-max values, which will require us to scale the numeric values in the preprocessing phase of our analysis.

In [23]:
```
train_df[['A2', 'A14']] = train_df[['A2', 'A14']] .astype(float)
```

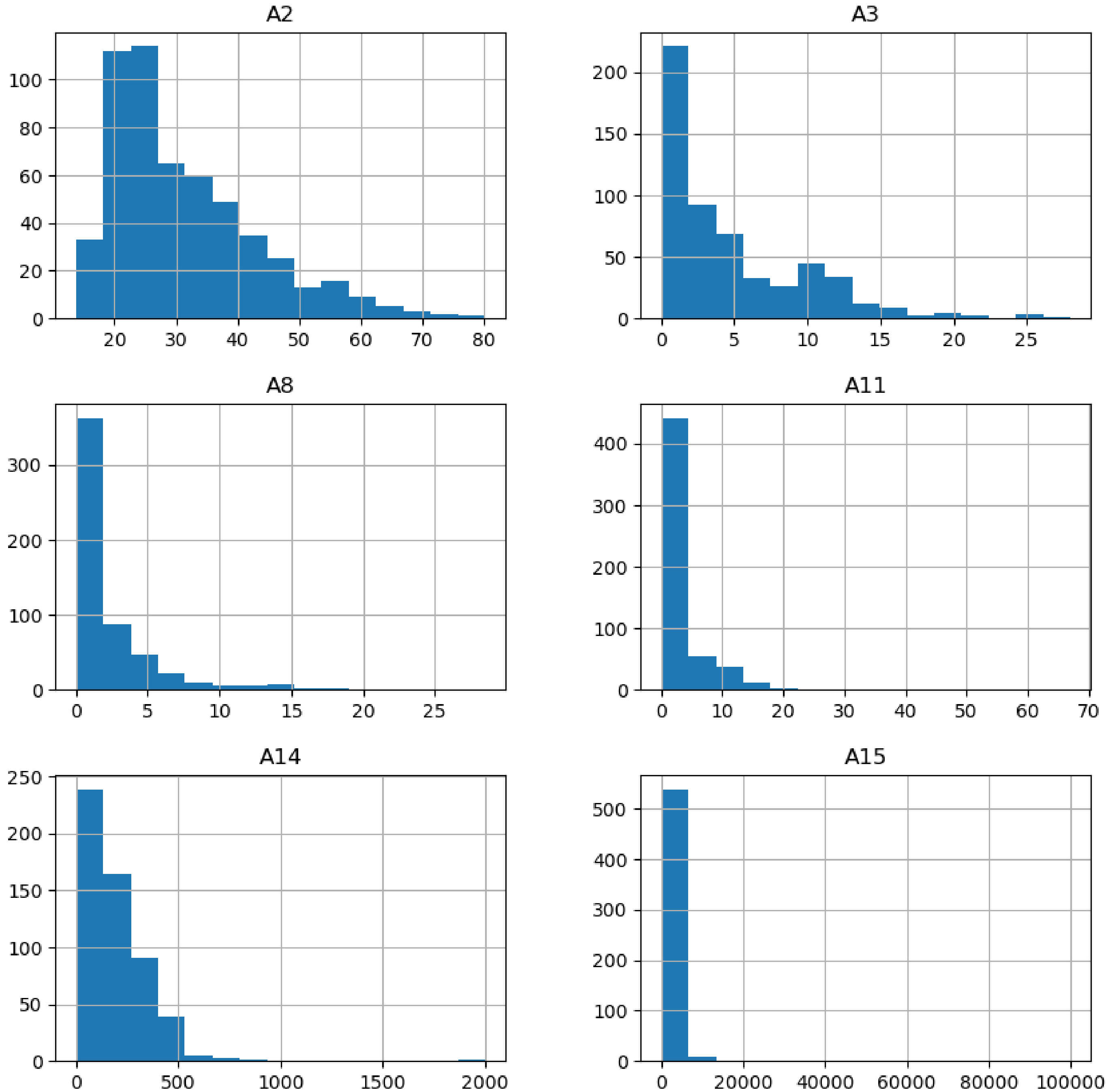
In [24]:
```
train_df.describe()
```

Out[24]:

	A2	A3	A8	A11	A14	A15
count	542.000000	552.000000	552.000000	552.000000	544.000000	552.000000
mean	31.210406	4.752745	2.211476	2.472826	182.981618	975.422101
std	11.938560	4.888587	3.329894	5.074328	166.134660	5553.903078
min	13.750000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	22.420000	1.040000	0.165000	0.000000	80.000000	0.000000
50%	27.670000	2.812500	1.000000	0.000000	160.000000	4.000000
75%	37.750000	7.155000	2.595000	3.000000	272.500000	369.000000
max	80.250000	28.000000	28.500000	67.000000	2000.000000	100000.000000

We also want to get a better understanding of the value distributions in the numeric columns by creating histograms with each of the values in the numeric columns.

In [25]:
```
hist = train_df.hist(bins = 15, figsize = (10,10))
```



Correlation matrix

Finally we want to look at the correlations among the numeric values. We can see that there are no major outliers in terms of correlations. This is a good sign, because it means that all the numeric features can potentially add value in the analysis.

In [26]:
```
corr = train_df.corr('spearman').style.background_gradient()
corr
```

Out[26]:

	A2	A3	A8	A11	A14	A15
A2	1.000000	0.093543	0.267358	0.115275	0.019440	0.037763
A3	0.093543	1.000000	0.261431	0.190279	-0.285688	0.109634
A8	0.267358	0.261431	1.000000	0.334260	-0.034651	0.092097
A11	0.115275	0.190279	0.334260	1.000000	-0.117220	0.424329
A14	0.019440	-0.285688	-0.034651	-0.117220	1.000000	-0.045223
A15	0.037763	0.109634	0.092097	0.424329	-0.045223	1.000000