State Farm Classification Coding Exercise

Part 2 - Test Data Set Exploratory Data Analysis and Feature Engineering

A. Import Libraries and Test Data Set, and Check for Missing Values

Import numpy and pandas.

In [1]:
```
import numpy as np
import pandas as pd
```

Import data visualization libraries and set %matplotlib inline.

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Import Exercise 2 test data set comma-separated (CSV) file into a Pandas dataframe.

In [3]:
```
test = pd.read_csv('../State_Farm/Data/exercise_02_test.csv', sep=',')
```

Create copy of test dataframe for exploratory data analysis and feature engineering.

In [4]:
```
test1 = test.copy()
```

View first five rows of test dataframe.

In [5]:
```
test.head()
```

Out[5]:

	x0	x1	x2	x3	x4	x5	x6	x7	
0	6.625366	54.479467	15.285444	-0.794648	22.498346	-29.212209	1.435134	-4.551934	5.9
1	3.796927	-20.244923	-18.084196	-1.113454	-3.551728	-4.025589	1.971885	-1.965186	13.2
2	31.875080	-61.467354	14.943580	0.979055	6.796937	-29.708041	4.778812	-2.682217	-17.1
3	15.266588	-18.454831	1.105534	-2.718771	-5.511702	2.252314	-8.017649	3.635776	-13.0
4	-17.616761	15.810515	-17.972025	-1.995724	-23.112552	-15.899861	-17.054154	4.097427	-7.7

5 rows × 100 columns

Obtain number of rows and columns in test dataframe.

In [6]:
```
test1.shape
```

Out[6]:
```
(10000, 100)
```

Check for presence of missing values for all features.

In [7]:
```
test1.isnull().sum().sort_values(ascending=False)
```

Out[7]:
```
x55    6
x5     5
x15    5
x87    5
x79    4
x3     4
x7     4
x94    4
x46    4
x48    4
x77    3
x74    3
x68    3
x62    3
x49    3
x52    3
x78    3
x42    3
x32    3
x13    3
x44    3
x0     3
x99    2
x89    2
x31    2
x21    2
x51    2
x24    2
x47    2
x88    2
      ..
x58    1
x73    1
x71    1
x70    1
x69    1
x67    1
x64    1
x63    1
x83    1
x59    1
x56    1
x76    1
x54    1
x2     1
x85    1
x98    1
x86    1
x95    1
x39    1
x8     1
x97    0
x4     0
x80    0
```

```
x25    0
x35    0
x43    0
x50    0
x53    0
x84    0
x38    0
Length: 100, dtype: int64
```

B. Explore and Engineer Numerical Features

Identify which test data set features are categorical.

In [8]:
```
test1.select_dtypes(exclude=['int64', 'float']).columns
```

Out[8]:
```
Index(['x34', 'x35', 'x41', 'x45', 'x68', 'x93'], dtype='object')
```

Check that the data types for all numerical features are float64.

In [9]:
```
num_features = test1.columns.difference(['x34', 'x35', 'x41', 'x45', 'x68', 'x93'
```

In [10]:
```
test1[num_features].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 94 columns):
x0     9997 non-null float64
x1     9998 non-null float64
x10    9998 non-null float64
x11    9998 non-null float64
x12    9999 non-null float64
x13    9997 non-null float64
x14    9998 non-null float64
x15    9995 non-null float64
x16    9998 non-null float64
x17    9999 non-null float64
x18    9999 non-null float64
x19    9999 non-null float64
x2     9999 non-null float64
x20    9998 non-null float64
x21    9998 non-null float64
x22    9999 non-null float64
x23    9999 non-null float64
x24    9998 non-null float64
x25    10000 non-null float64
x26    9998 non-null float64
x27    9999 non-null float64
x28    9999 non-null float64
x29    9999 non-null float64
x3     9996 non-null float64
x30    9999 non-null float64
x31    9998 non-null float64
x32    9997 non-null float64
x33    9998 non-null float64
x36    9998 non-null float64
x37    9998 non-null float64
x38    10000 non-null float64
x39    9999 non-null float64
x4     10000 non-null float64
x40    9998 non-null float64
x42    9997 non-null float64
x43    10000 non-null float64
x44    9997 non-null float64
x46    9996 non-null float64
x47    9998 non-null float64
x48    9996 non-null float64
x49    9997 non-null float64
x5     9995 non-null float64
x50    10000 non-null float64
x51    9998 non-null float64
x52    9997 non-null float64
x53    10000 non-null float64
x54    9999 non-null float64
x55    9994 non-null float64
x56    9999 non-null float64
x57    9998 non-null float64
x58    9999 non-null float64
```

```
x59    9999 non-null float64
x6     9999 non-null float64
x60    9998 non-null float64
x61    9998 non-null float64
x62    9997 non-null float64
x63    9999 non-null float64
x64    9999 non-null float64
x65    9998 non-null float64
x66    9998 non-null float64
x67    9999 non-null float64
x69    9999 non-null float64
x7     9996 non-null float64
x70    9999 non-null float64
x71    9999 non-null float64
x72    9998 non-null float64
x73    9999 non-null float64
x74    9997 non-null float64
x75    9998 non-null float64
x76    9999 non-null float64
x77    9997 non-null float64
x78    9997 non-null float64
x79    9996 non-null float64
x8     9999 non-null float64
x80    10000 non-null float64
x81    9998 non-null float64
x82    9998 non-null float64
x83    9999 non-null float64
x84    10000 non-null float64
x85    9999 non-null float64
x86    9999 non-null float64
x87    9995 non-null float64
x88    9998 non-null float64
x89    9998 non-null float64
x9     9998 non-null float64
x90    9999 non-null float64
x91    9999 non-null float64
x92    9999 non-null float64
x94    9996 non-null float64
x95    9999 non-null float64
x96    9998 non-null float64
x97    10000 non-null float64
x98    9999 non-null float64
x99    9998 non-null float64
dtypes: float64(94)
memory usage: 7.2 MB
```

View scatter matrix of numerical features to inspect their distributions.

In [11]:
```
test1[num_features].hist(figsize=(20,16));
```

- All the numerical features are normally distributed. The number of missing values for each feature ranges from 1 to 6 while the total number of rows in the test data set is 10,000. Given these conditions, I decided to impute the missing values with the mean of the feature.

Impute missing values in numerical features with mean.

In [12]:
```
test2 = test1.fillna(test1[num_features].mean())
```

Check numerical features for any missing values.

In [13]:
```
test2[num_features].isnull().sum().sort_values(ascending=False)
```

Out[13]:
```
x99    0
x42    0
x31    0
x32    0
x33    0
x36    0
x37    0
x38    0
x39    0
x4     0
x40    0
x43    0
x3     0
x44    0
x46    0
x47    0
x48    0
x49    0
x5     0
x50    0
x51    0
x52    0
x30    0
x29    0
x98    0
x18    0
x1     0
x10    0
x11    0
x12    0
      ..
x91    0
x92    0
x94    0
x95    0
x96    0
x97    0
x78    0
x77    0
x76    0
x75    0
x57    0
x58    0
x59    0
x6     0
x60    0
x61    0
x62    0
x63    0
x64    0
x65    0
x66    0
x67    0
x69    0
```

```
x7     0
x70    0
x71    0
x72    0
x73    0
x74    0
x0     0
Length: 94, dtype: int64
```

View scatter matrix of imputed numerical features to check if the mean imputations skewed their distributions.

In [14]:
```
test2[num_features].hist(figsize=(20,16));
```

- The histograms for all the numerical features show that their distributions still continue to remain normal after imputing their missing values with the mean.

C. Explore and Engineer Categorical Features

Check categorical feature data types.

In [15]:
```
cat_features1 = ['x34', 'x35', 'x41', 'x45', 'x68', 'x93']
```

In [16]:
```
test2[cat_features1].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
x34    9998 non-null object
x35    10000 non-null object
x41    9998 non-null object
x45    9998 non-null object
x68    9997 non-null object
x93    9999 non-null object
dtypes: object(6)
memory usage: 468.8+ KB
```

View summary statistics for categorical features.

In [17]:
```
test2.describe(include=['object'])
```

Out[17]:

	x34	x35	x41	x45	x68	x93
count	9998	10000	9998	9998	9997	9999
unique	10	8	9861	10	12	3
top	volkswagon	wed	$-235.5	0.01%	July	asia
freq	3234	3745	3	2428	2775	8868

Convert currency and percent string features (x41 and x45) to float data type.

In [18]:
```
test2[['x41_flt']] = test2[['x41']].apply(lambda x: x.str.replace('$','')).astype
test2[['x45_pct']] = test2[['x45']].apply(lambda x: x.str.replace('%','')).astype
test3 = test2.drop(['x41', 'x45'], axis=1)
```

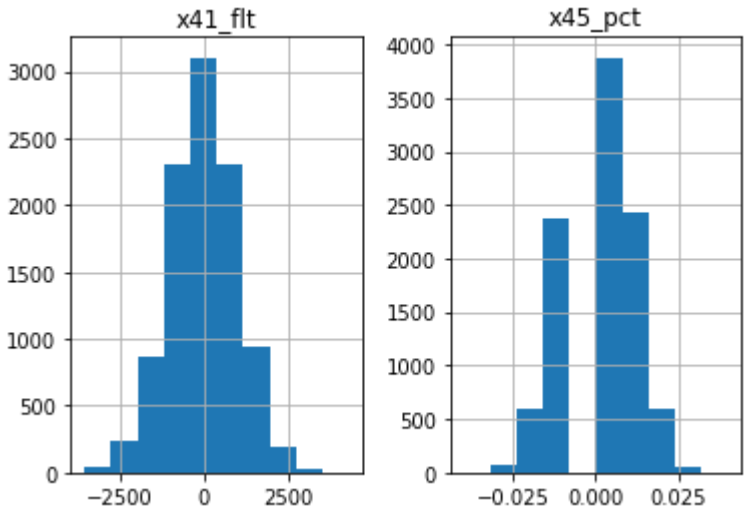
Check the number of missing values for the numerical x41 and x45 features.

In [19]:
```
test3[['x41_flt', 'x45_pct']].isnull().sum()
```

Out[19]:
```
x41_flt    2
x45_pct    2
dtype: int64
```

View scatter matrix of numerical x41 and x45 features to inspect their distributions.

In [20]:
```
test3[['x41_flt', 'x45_pct']].hist();
```



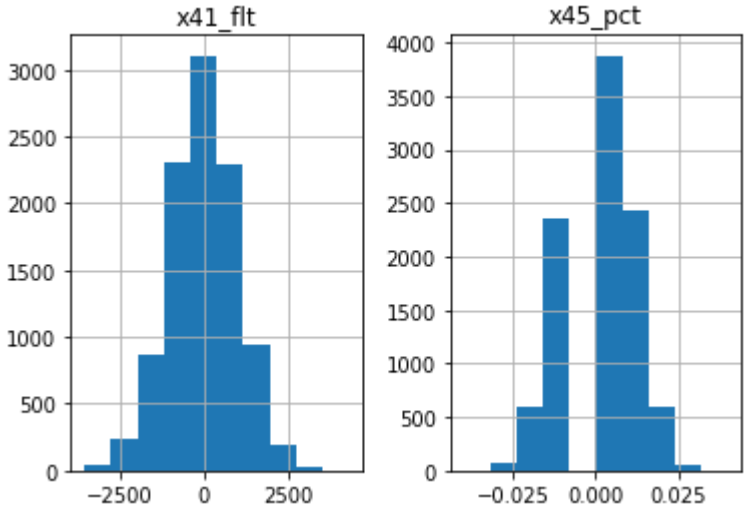
- The numerical x41 and x45 features are normally distributed. The number of missing values for the numerical x41 and x45 features is 2 and 2, respectively. Again, the total number of rows in the test data set is 10,000. Given these conditions, I decided to impute the missing values with the mean of the feature.

Impute missing values in numerical x41 and x45 features with mean.

In [21]:
```
test4 = test3.fillna(test3[['x41_flt', 'x45_pct']].mean())
```

View scatter matrix of imputed numerical x41 and x45 features to check if the mean imputations skewed their distributions.

In [22]:
```
test4[['x41_flt', 'x45_pct']].hist();
```



- The histograms for the numerical x41 and x45 features show that their distributions still continue to remain normal after imputing their missing values with the mean.

Check for features that still have missing values.

In [23]:
```
test4.isnull().sum().sort_values(ascending=False).head()
```

Out[23]:
```
x68        3
x34        2
x93        1
x45_pct    0
x35        0
dtype: int64
```

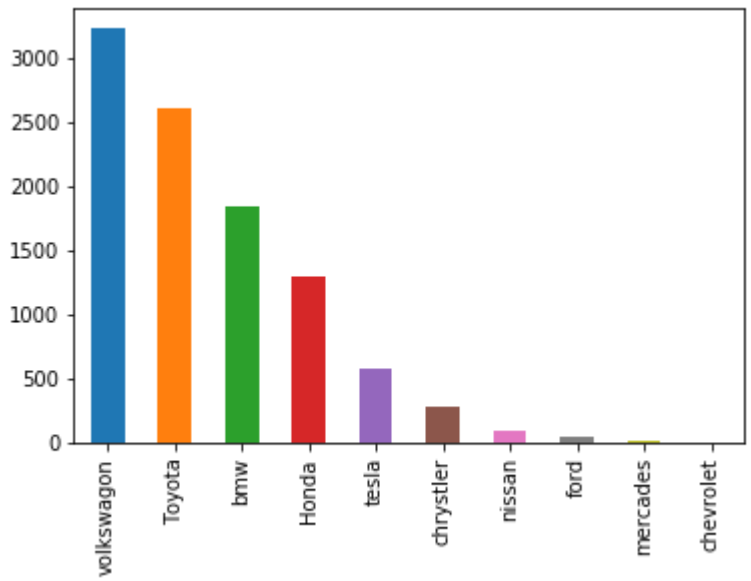
Identify remaining categorical features.

In [24]:
```
test4.select_dtypes(exclude=['int64', 'float']).columns
```

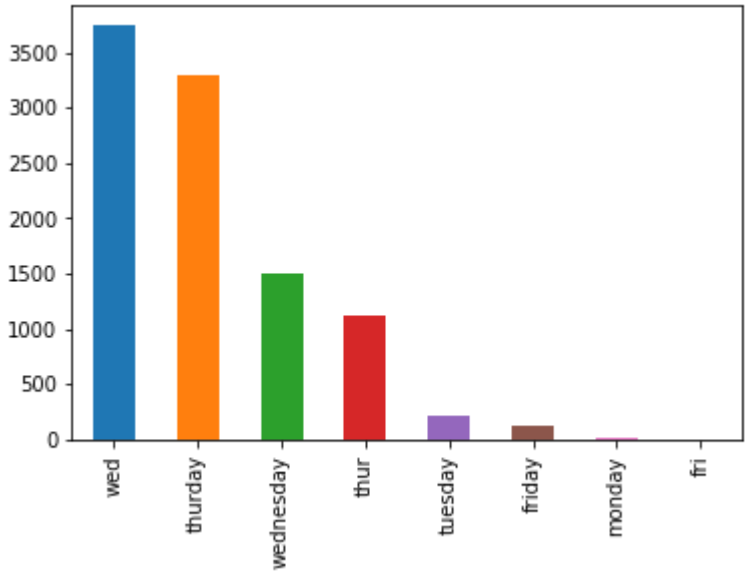
Out[24]:
```
Index(['x34', 'x35', 'x68', 'x93'], dtype='object')
```

View bar plots for categorical features of x34, x35, x68, and x93.

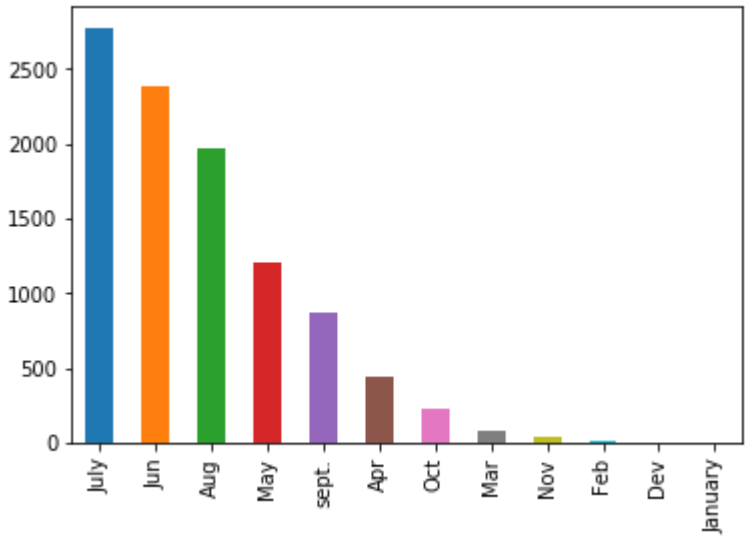
In [25]:
```
test4.x34.value_counts().plot(kind='bar');
```



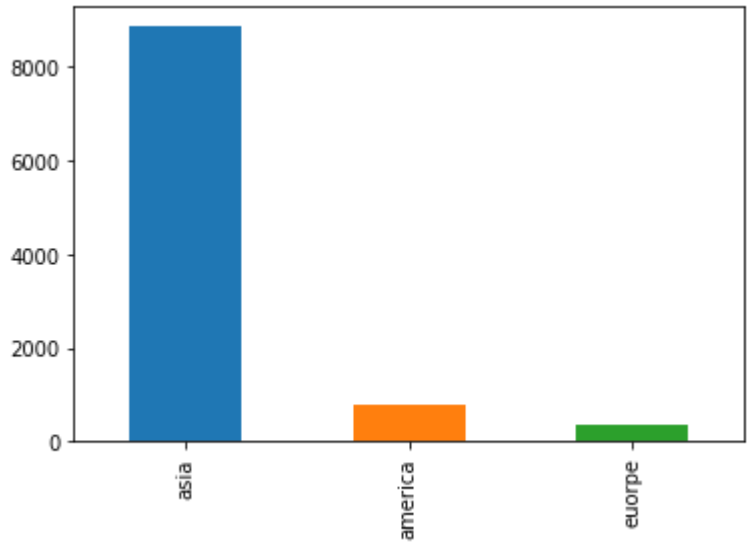
In [26]:
```
test4.x35.value_counts().plot(kind='bar');
```



In [27]:
```
test4.x68.value_counts().plot(kind='bar');
```



In [28]:
```
test4.x93.value_counts().plot(kind='bar');
```



- The missing values for the categorical features of x34, x35, x68, and x93 are truly blank. In other words, much more domain knowledge is required to impute these missing values. Going forward, I will assign these missing values their own missing category.

Replace all categorical feature missing values with their own missing category.

In [29]:
```
test4['x34'] = test4.x34.fillna('No_Car_Make')
test4['x68'] = test4.x68.fillna('No_Month')
test4['x93'] = test4.x93.fillna('No_Continent')
```

Check that all categorical features have zero missing values.

In [30]:
```
test4[['x34', 'x35', 'x68', 'x93']].isnull().sum().sort_values(ascending=False)
```

Out[30]:
```
x93    0
x68    0
x35    0
x34    0
dtype: int64
```

Obtain value counts for each x34 category.

In [31]:
```
test4.x34.value_counts()
```

Out[31]:
```
volkswagon     3234
Toyota         2616
bmw            1844
Honda          1293
tesla           583
chrystler       281
nissan           95
ford             40
mercades         10
chevrolet         2
No_Car_Make       2
Name: x34, dtype: int64
```

Clean x34 feature car make names and obtain value counts again.

In [32]:
```
test4['x34'] = test4.x34.map({'volkswagon':'Volkswagen', 'Toyota':'Toyota', 'bmw'
                              'chrystler':'Chrysler', 'nissan':'Nissan', 'ford':'F
                               'chevrolet':'Chevrolet', 'No_Car_Make':'No_Car_Make

test4.x34.value_counts()
```

Out[32]:
```
Volkswagen     3234
Toyota         2616
BMW            1844
Honda          1293
Tesla           583
Chrysler        281
Nissan           95
Ford             40
Mercedes         10
Chevrolet         2
No_Car_Make       2
Name: x34, dtype: int64
```

Create x34 dummy features with Volkswagen as reference category and add it to test dataframe.

In [33]:
```
x34_dummies = pd.get_dummies(test4.x34).drop('Volkswagen', axis=1)
test5 = pd.concat([test4, x34_dummies], axis=1)
```

Obtain value counts for each x35 category.

In [34]:
```
test5.x35.value_counts()
```

Out[34]:
```
wed          3745
thurday      3285
wednesday    1496
thur         1117
tuesday       218
friday        121
monday         13
fri             5
Name: x35, dtype: int64
```

Clean x35 feature weekday names and obtain value counts again.

In [35]:
```
test5['x35'] = test5.x35.map({'wed':'Wednesday', 'thurday':'Thursday', 'wednesday
                              'tuesday':'Tuesday', 'friday':'Friday', 'monday':'M
test5.x35.value_counts()
```

Out[35]:
```
Wednesday    5241
Thursday     4402
Tuesday       218
Friday        126
Monday         13
Name: x35, dtype: int64
```

Create x35 dummy features with Wednesday as reference category and add it to test dataframe.

In [36]:
```
x35_dummies = pd.get_dummies(test5.x35).drop('Wednesday', axis=1)
test6 = pd.concat([test5, x35_dummies], axis=1)
```

Obtain value counts for each x68 category.

In [37]:
```
test6.x68.value_counts()
```

Out[37]:
```
July        2775
Jun         2385
Aug         1964
May         1208
sept.        868
Apr          437
Oct          228
Mar           82
Nov           36
Feb            6
Dev            5
January        3
No_Month       3
Name: x68, dtype: int64
```

Clean x68 feature month names and obtain value counts again.

In [38]:
```
test6['x68'] = test6.x68.map({'July':'July', 'Jun':'June', 'Aug':'August', 'May':
                              'Oct':'October', 'Mar':'March', 'Nov':'November', '
                              'January':'January', 'No_Month':'No_Month'})
test6.x68.value_counts()
```

Out[38]:
```
July         2775
June         2385
August       1964
May          1208
September     868
April         437
October       228
March          82
November       36
February        6
December        5
January         3
No_Month        3
Name: x68, dtype: int64
```

Create x68 dummy features with July as reference category and add it to test dataframe.

In [39]:
```
x68_dummies = pd.get_dummies(test6.x68).drop('July', axis=1)
test7 = pd.concat([test6, x68_dummies], axis=1)
```

Obtain value counts for each x93 category.

In [40]:
```
test7.x93.value_counts()
```

Out[40]:
```
asia           8868
america         788
euorpe          343
No_Continent      1
Name: x93, dtype: int64
```

Clean x93 feature continent names and obtain value counts again.

In [41]:
```
test7['x93'] = test7.x93.map({'asia':'Asia', 'america':'America', 'euorpe':'Europ
test7.x93.value_counts()
```

Out[41]:
```
Asia           8868
America         788
Europe          343
No_Continent      1
Name: x93, dtype: int64
```

Create x93 dummy features with Asia as reference category and add it to test dataframe.

In [42]:
```
x93_dummies = pd.get_dummies(test7.x93).drop('Asia', axis=1)
test8 = pd.concat([test7, x93_dummies], axis=1)
```

D. Finalize and Export Cleaned Test Data Set for Export

Drop categorical features from test dataframe.

In [43]:
```
test_cleaned = test8.drop(['x34', 'x35', 'x68', 'x93'], axis=1)
```

Obtain number of rows and columns in test dataframe with engineered and cleaned features.

In [44]:
```
test_cleaned.shape
```

Out[44]:
```
(10000, 125)
```

Check for any remaining missing values in test dataframe with engineered and cleaned features.

In [45]:
```
test_cleaned.isnull().sum().sort_values(ascending=False)
```

Out[45]:
```
No_Continent    0
x49             0
x32             0
x33             0
x36             0
x37             0
x38             0
x39             0
x40             0
x42             0
x43             0
x44             0
x46             0
x47             0
x48             0
x50             0
x65             0
x51             0
x52             0
x53             0
x54             0
x55             0
x56             0
x57             0
x58             0
x59             0
x60             0
x61             0
x62             0
x63             0
               ..
x98             0
x97             0
x69             0
x70             0
x71             0
x72             0
x73             0
x74             0
x75             0
x76             0
x77             0
x78             0
x79             0
x80             0
x81             0
x82             0
x83             0
x84             0
x85             0
x86             0
x87             0
x88             0
x89             0
```

```
x90             0
x91             0
x92             0
x94             0
x95             0
x96             0
x0              0
Length: 125, dtype: int64
```

Export test dataframe with engineered and cleaned features to CSV file.

In [46]:
```
test_cleaned.to_csv('../State_Farm/Data/test_cleaned.csv', sep=',', index=False)
```

Save test dataframe with engineered and cleaned features to pickle file for models to make predictions on.

In [47]:
```
test_cleaned.to_pickle('../State_Farm/Data/test_cleaned.pickle')
```