Data_Analytics_Project

January 23, 2019

Table of Contents

1 Introduction: Data Analytics Project

In this notebook, I will demonstrate how to implement a data analytics project, focusing on data profiling, data engineering and machine learning techniques. I will go through the entire data analytics process including loading & manipulating data, profiling the data, exploring it to find trends, generating traning & testing data to be ready for machine learning use, establishing a baseline model, evaluating and comparing several machine learning methods, interpreting the results, and presenting the results.

1.1 Dataset

I am using the data on Vehicle Management System (VMS) for three country projects, stored at csv format files. The data includes collected features of vehicle usage, such as date of vehicle use, destination, kilometer at beginning of a trip, kilometer at the end of a trip, fuel added and payment method (cash or other) to purchase fuel. The objective is to study what **top 3** vehicle usage factors or what patterns being specific to the country projects by using **supervised, regression technique**, which will be learning a mapping from the features in a set of training data with known labels to the target (the label) in this case the countries.

1.2 Python Library

I will use Python library **pandas** & **numpy** to read-in and process the data from a local machine, **pandas_profiling** to profile the data, **matplotlib** & **seaborn** to visualize the data and machine learning results, **scipy** for statistical analysis and **scikit-learn** for machine learning regression approaches. For the machine learning regression approaches, I will examine and evaluate **Linear Regression**, **ElasticNet Regression**, **Random Forest**, **Extra Trees**, **SVM** and **Gradient Boosted** approaches.

```
In [1]: # Pandas and numpy for data processing
        import pandas as pd
        import numpy as np
        # Make the random numbers predictable
        np.random.seed(42)

        # pandas-profiling for data profiling and quality assessment
        import pandas_profiling

In [2]: # Scipy for stats analysis
        import scipy
        from scipy import stats

In [3]: # Standard ML Models for comparison
        from sklearn.linear_model import LinearRegression
        from sklearn.linear_model import ElasticNet
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.ensemble import ExtraTreesRegressor
        from sklearn.ensemble import GradientBoostingRegressor
        from sklearn.svm import SVR
```

```
# Splitting data into training/testing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# Metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error, median_absolute_e
```

In [4]:
```
# Matplotlib and seaborn for visualization
import matplotlib.pyplot as plt
import matplotlib
import seaborn as sns

# Set up matplotlib environment
%matplotlib inline
matplotlib.rcParams['font.size'] = 16
matplotlib.rcParams['figure.figsize'] = (9, 9)

from IPython.core.pylabtools import figsize
```

2 Exploratory Data Loading

2.1 Data Loading

2.1.1 Read in data from csv files

In [5]:
```
# Read three data sets
# need a different encoding rather than the default utf-8
# due to some specific letters in the data sets
df1 = pd.read_csv('../datasets/1_Log.csv', encoding = "ISO-8859-1")
df2 = pd.read_csv('../datasets/2_Log.csv', encoding = "ISO-8859-1")
df3 = pd.read_csv('../datasets/3_Log.csv', encoding = "ISO-8859-1")
```

The data dictionary for the columns is: * ID : Vehicle Usage History System ID (data type: non-null int64) * Date : Vehicle Usage Date (data type: non-null object) * Country : Vehicle Usage Location/Country ID (data type: non-null int64) * VehID : Vehicle System ID (data type: non-null int64) * Destination : Vehicle Driving Stops and Final Destination (data type: non-null object) * KmInit : Vehicle Usage Initial Kilometer Each Time (data type: non-null int64) * KmFinal : Vehicle Usage Final Kilometer Each Time (data type: non-null int64) * FuelBought : How Much Fuel Was Bought (data type: non-null float64) * AmountFuel : How Much Was Paid to Buy Fuel (data type: non-null float64) * CashFuel : How Much Fuel Was Bought By Cash (data type: non-null float64) * AmountCash : How Much Cash Was Paid to Buy Fuel (data type: non-null float64) * FuelKm : How Many Kilometer (after initial kilometer) When Fuel Was Bought (data type: non-null int64) * DriverID : Driver System ID (data type: non-null int64)

In [6]:
```
df1.head(10)
```

Out[6]:
```
   ID       Date  Country  VehID               Destination  KmInit  \
0  30  1/11/2008        1     12         Nairobi - Amboseli   67029
1  31  1/14/2008        1     12           Tsavo - Nairobi   67429
```

```
2  32  1/14/2008         1      12              CRS - Westlands - CRS   67848
3  33  1/17/2008         1      12   Nairobi - Machakos - Within Town   67850
4  34  1/19/2008         1      12                  Machakos - Nairobi   68072
5  36  1/22/2008         1      12                    Nairobi - Nakuru   68186
6  37  1/23/2008         1      12                Kakuru - Within Town   68418
7  38  1/24/2008         1      12           Nakuru - Mogotio - Nakuru   68501
8  39  1/26/2008         1      12  Nakuru - IDP Camps within  Nakuru   68737
9  74   1/6/2008         1      35                   Nairobi - nairobi   49180

   KmFinal  FuelBought  AmountFuel  CashFuel  AmountCash  FuelKm  DriverID
0    67429    0.000000         0.0       0.0         0.0       0         1
1    67848    0.000000         0.0       0.0         0.0       0         1
2    67850    0.000000         0.0       0.0         0.0       0         1
3    68072    0.000000         0.0       0.0         0.0       0         1
4    68186    0.000000         0.0       0.0         0.0       0         1
5    68418  110.220001      8500.0       0.0         0.0   68190         1
6    68501    0.000000         0.0       0.0         0.0       0         1
7    68737    0.000000         0.0       0.0         0.0       0         1
8    69389    0.000000         0.0       0.0         0.0       0         1
9    49219   34.380001      3015.0       0.0         0.0   49200       202
```

```
In [7]: df2.head(10)
```

```
Out[7]:    ID        Date  Country  VehID Destination  KmInit  KmFinal  FuelBought  \
        0   4   7/27/2011        2      8         CEV  124535   124542    0.000000
        1   5   7/28/2011        2      8         CEV  124542   124765    0.000000
        2   6   8/23/2011        2      8         CEV  124765   124777   41.849998
        3   7   8/27/2011        2      8         CEV  124777   124931    0.000000
        4   8   9/15/2011        2      8         CEV  124931   124943    0.000000
        5   9  10/30/2011        2      8         CEV  125062   125132    0.000000
        6  11  10/30/2011        2      8         CEV  125132   125152    0.000000
        7  12  10/30/2011        2      8         CEV  125152   125163    0.000000
        8  13  10/30/2011        2      8         CEV  125163   125171    0.000000
        9  15  10/25/2011        2      8         cev  124943   125062    0.000000

           AmountFuel  CashFuel  AmountCash  FuelKm  DriverID
        0         0.0       0.0         0.0       0         3
        1         0.0       0.0         0.0       0         1
        2    129310.0       0.0         0.0  124773         1
        3         0.0       0.0         0.0       0         1
        4         0.0       0.0         0.0       0         1
        5         0.0       0.0         0.0       0         1
        6         0.0       0.0         0.0       0         1
        7         0.0       0.0         0.0       0         1
        8         0.0       0.0         0.0       0         1
        9         0.0       0.0         0.0       0         1
```

```
In [8]: df3.head(10)
```

```
Out[8]:    ID        Date  Country  VehID      Destination  KmInit  KmFinal  \
        0   4    2/1/2008        3      1     Wuse Austoma  148405   148462
        1   6   1/30/2008        3      4             wuse   66768    66798
        2   7   1/30/2008        3      4           wuse 2   66798    66813
        3   8   1/31/2008        3      4         Air port   66813    66930
        4   9   1/31/2008        3      4    wuse personal   66930    66962
        5  10   1/31/2008        3      4  wuse school run   66962    66991
        6  11    2/2/2008        3      4     wuse pick up   66991    67005
        7  12    2/2/2008        3      4    wuse  pick up   67005    67020
        8  13    2/2/2008        3      4  usaid aso drive   67020    67033
        9  17    2/1/2008        3      4     FUEL AUSTOMA   67033    67043

           FuelBought   AmountFuel     CashFuel   AmountCash  FuelKm  DriverID
        0   16.030001  1122.099976     0.000000     0.000000       0       139
        1    0.000000     0.000000     0.000000     0.000000       0       137
        2    0.000000     0.000000     0.000000     0.000000       0       137
        3    0.000000     0.000000     0.000000     0.000000       0       139
        4    0.000000     0.000000     0.000000     0.000000       0       148
        5    0.000000     0.000000     0.000000     0.000000       0       137
        6    0.000000     0.000000     0.000000     0.000000       0       137
        7    0.000000     0.000000     0.000000     0.000000       0       145
        8    0.000000     0.000000     0.000000     0.000000       0       143
        9   52.980000  3708.600098  3708.600098  3708.600098      10       143
```

2.2 Data Set Information

```
In [9]: print("First data set - df1: ", df1.shape)
        print("Second data set - df2: ", df2.shape)
        print("Third data set - df3: ", df3.shape)
        print("-----------------------------------")
        print(df1.info(verbose=True))
        print("-----------------------------------")
        print(df2.info(verbose=True))
        print("-----------------------------------")
        print(df3.info(verbose=True))
```

```
First data set - df1:  (64544, 13)
Second data set - df2:  (61199, 13)
Third data set - df3:  (93447, 13)
-----------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 64544 entries, 0 to 64543
Data columns (total 13 columns):
ID             64544 non-null int64
Date           64544 non-null object
Country        64544 non-null int64
VehID          64544 non-null int64
Destination    63601 non-null object
```

```
KmInit         64544 non-null int64
KmFinal        64544 non-null int64
FuelBought     64544 non-null float64
AmountFuel     64544 non-null float64
CashFuel       64544 non-null float64
AmountCash     64544 non-null float64
FuelKm         64544 non-null int64
DriverID       64544 non-null int64
dtypes: float64(4), int64(7), object(2)
memory usage: 6.4+ MB
None
-----------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61199 entries, 0 to 61198
Data columns (total 13 columns):
ID             61199 non-null int64
Date           61199 non-null object
Country        61199 non-null int64
VehID          61199 non-null int64
Destination    55021 non-null object
KmInit         61199 non-null int64
KmFinal        61199 non-null int64
FuelBought     61199 non-null float64
AmountFuel     61199 non-null float64
CashFuel       61199 non-null float64
AmountCash     61199 non-null float64
FuelKm         61199 non-null int64
DriverID       61199 non-null int64
dtypes: float64(4), int64(7), object(2)
memory usage: 6.1+ MB
None
-----------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 93447 entries, 0 to 93446
Data columns (total 13 columns):
ID             93447 non-null int64
Date           93447 non-null object
Country        93447 non-null int64
VehID          93447 non-null int64
Destination    93416 non-null object
KmInit         93447 non-null int64
KmFinal        93447 non-null int64
FuelBought     93447 non-null float64
AmountFuel     93447 non-null float64
CashFuel       93447 non-null float64
AmountCash     93447 non-null float64
FuelKm         93447 non-null int64
DriverID       93447 non-null int64
```

```
dtypes: float64(4), int64(7), object(2)
memory usage: 9.3+ MB
None
```

2.2.1 Describe for numeric columns

In [10]: `df1.describe()`

```
Out[10]:                 ID  Country          VehID         KmInit        KmFinal  \
         count  64544.000000  64544.0   64544.000000   64544.000000   64544.000000
         mean   60476.359971      1.0     156.237605   68860.702265   68931.561152
         std    24608.652709      0.0     112.221037   53398.297490   53425.787428
         min       30.000000      1.0       1.000000       0.000000      41.000000
         25%    45583.750000      1.0      53.000000   29811.750000   29867.750000
         50%    62757.500000      1.0      96.000000   54676.500000   54730.000000
         75%    80255.250000      1.0     243.000000   94024.250000   94110.250000
         max    97199.000000      1.0     336.000000  287891.000000  288029.000000

                  FuelBought     AmountFuel      CashFuel    AmountCash         FuelKm  \
         count  64544.000000   64544.000000  64544.000000  64544.000000   64544.000000
         mean       8.094475     786.338400      0.107624     11.837927   10031.041801
         std       24.973910    2691.563365      2.776170    308.006551   34048.292617
         min        0.000000       0.000000      0.000000      0.000000       0.000000
         25%        0.000000       0.000000      0.000000      0.000000       0.000000
         50%        0.000000       0.000000      0.000000      0.000000       0.000000
         75%        0.000000       0.000000      0.000000      0.000000       0.000000
         max      244.429993  181930.000000    151.000000  18120.000000  379475.000000

                    DriverID
         count  64544.000000
         mean     460.762023
         std      322.146166
         min        1.000000
         25%      208.000000
         50%      468.000000
         75%      633.000000
         max     1117.000000
```

In [11]: `df2.describe()`

```
Out[11]:                 ID  Country          VehID         KmInit        KmFinal  \
         count  61199.000000  61199.0   61199.000000   61199.000000   61199.000000
         mean   31701.482246      2.0      33.191801   94926.993072   95007.556545
         std    18386.230520      0.0      22.245209   71295.990304   71300.943003
         min        4.000000      2.0       2.000000       0.000000      20.000000
         25%    15793.500000      2.0      13.000000   31914.000000   31960.000000
         50%    31461.000000      2.0      27.000000   80322.000000   80398.000000
         75%    47662.500000      2.0      54.000000  150082.000000  150138.000000
```

```
max    63621.000000      2.0     88.000000  320376.000000  320390.000000

         FuelBought    AmountFuel      CashFuel     AmountCash        FuelKm  \
count  61199.000000  6.119900e+04  61199.000000   61199.000000  61199.000000
mean       9.569300  2.677606e+04      0.397249    1088.405013  14385.406281
std       26.852828  7.483434e+04      5.450899   15032.425844  44156.859173
min        0.000000  0.000000e+00      0.000000       0.000000      0.000000
25%        0.000000  0.000000e+00      0.000000       0.000000      0.000000
50%        0.000000  0.000000e+00      0.000000       0.000000      0.000000
75%        0.000000  0.000000e+00      0.000000       0.000000      0.000000
max      529.750000  2.190091e+06    186.800003  508000.000000 999939.000000

           DriverID
count  61199.000000
mean      21.228435
std       25.715557
min        1.000000
25%        1.000000
50%       13.000000
75%       27.000000
max       97.000000
```

In [12]: df3.describe()

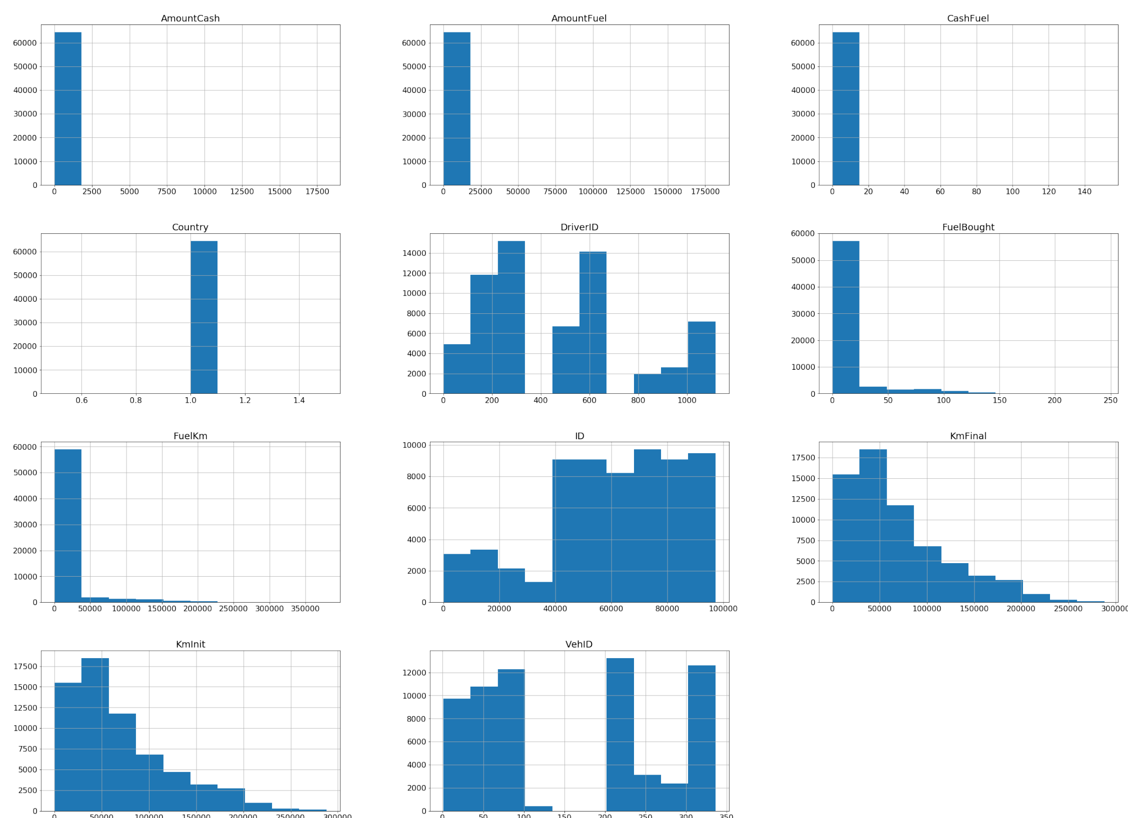
```
Out[12]:
                 ID  Country         VehID         KmInit        KmFinal  \
count  93447.000000  93447.0  93447.000000   93447.000000   93447.000000
mean   48834.597173      3.0     40.740837   70418.718707   70471.444423
std    28858.653297      0.0     37.015758   54364.011870   54367.940770
min        4.000000      3.0      1.000000      15.000000      25.000000
25%    23813.500000      3.0     18.000000   23727.000000   23769.000000
50%    48094.000000      3.0     26.000000   61176.000000   61217.000000
75%    73416.500000      3.0     54.000000  104733.500000  104811.500000
max    99632.000000      3.0    158.000000  232066.000000  232101.000000

         FuelBought    AmountFuel      CashFuel    AmountCash         FuelKm  \
count  93447.000000  9.344700e+04  93447.000000  93447.000000   93447.000000
mean       6.186339  6.961235e+02      3.501538    279.907707    3257.298693
std      224.156646  2.276397e+04    217.021757   1434.464042   19505.136901
min        0.000000  0.000000e+00      0.000000      0.000000       0.000000
25%        0.000000  0.000000e+00      0.000000      0.000000       0.000000
50%        0.000000  0.000000e+00      0.000000      0.000000       0.000000
75%        0.000000  0.000000e+00      0.000000      0.000000       0.000000
max    68050.000000  6.762700e+06  66015.000000  95566.898440  312965.000000

          DriverID
count  93447.00000
mean     373.89272
std      316.55336
```

```
min         1.00000
25%       144.00000
50%       262.00000
75%       392.00000
max      2125.00000
```

2.2.2 Histogram of columns

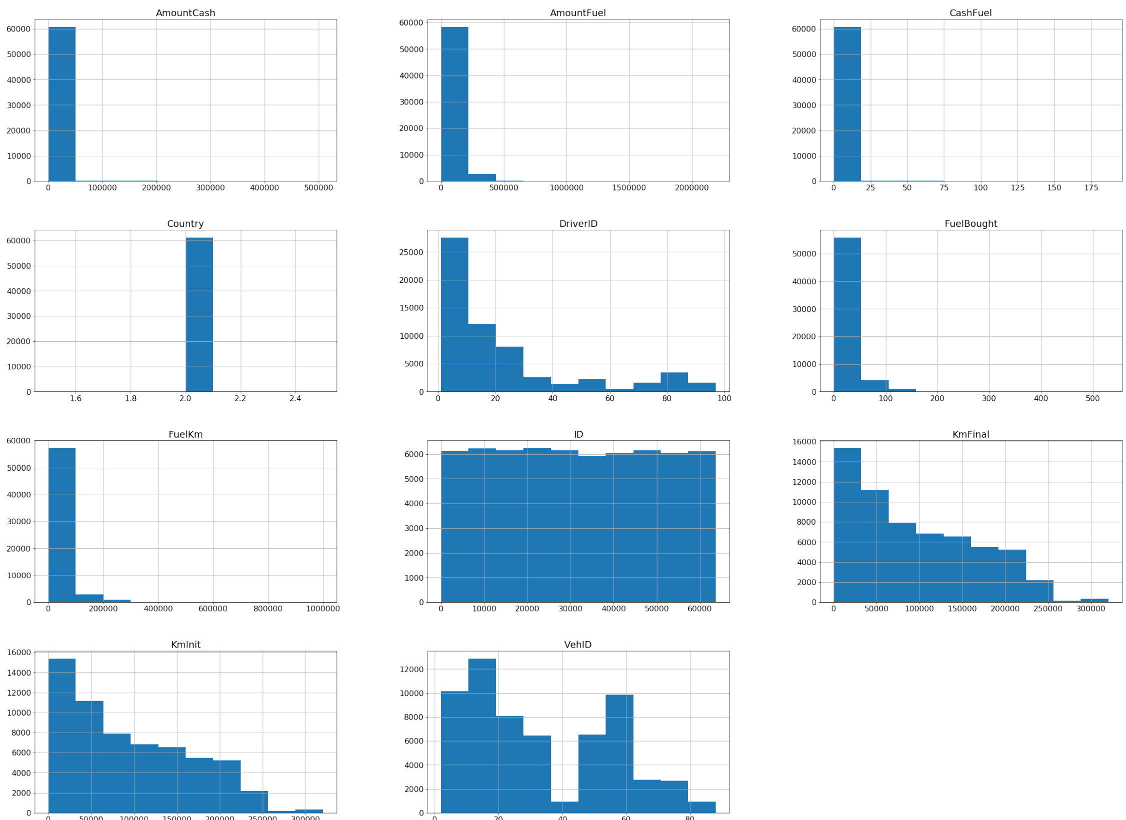
In [13]:
```
df1.hist(figsize=(40, 30))
```

Out[13]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3f8f5668>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3f8b9fd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3f869240>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3f8124a8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3f83b588>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3f7e37f0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3f80ba58>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3f7b3cf8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3f7b3d30>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3f78f1d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3f738438>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3f6e06a0>]],
      dtype=object)
```



```
In [14]: df2.hist(figsize=(40, 30))
```

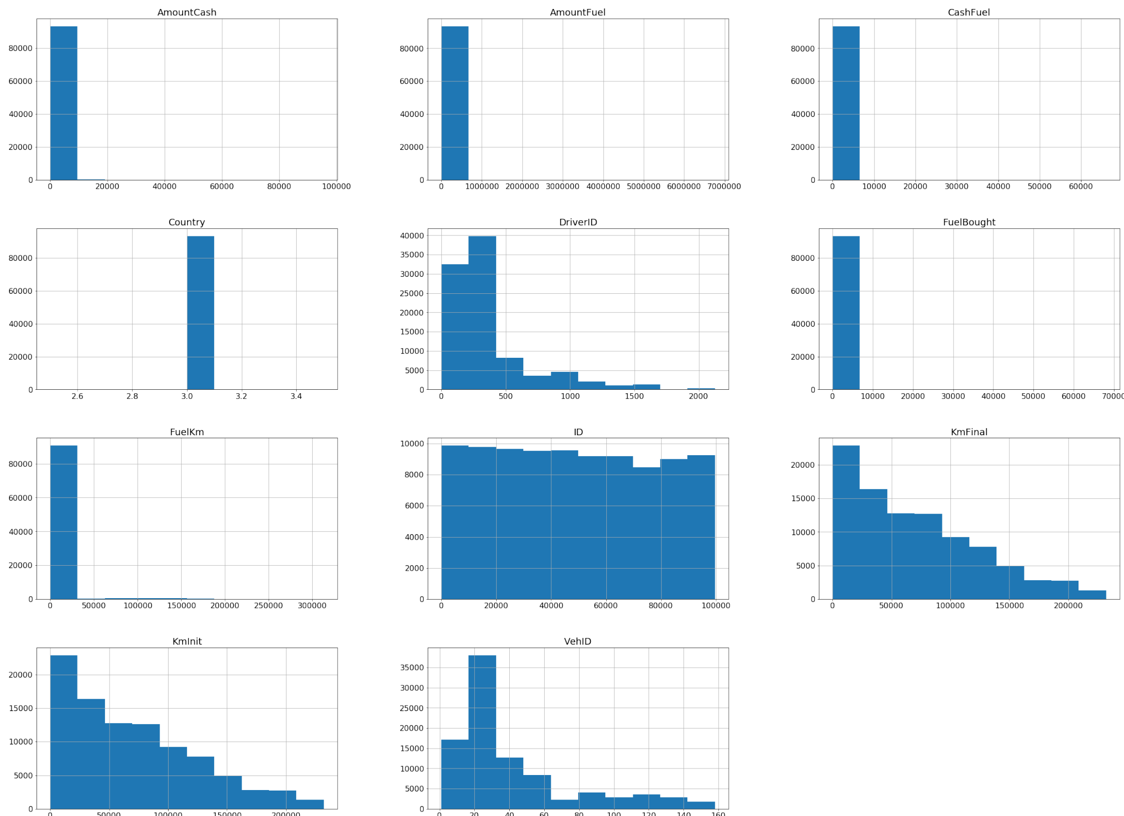
```
Out[14]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3f486cc0>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3de09080>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3de275c0>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3ddcdb38>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3dd810b8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3dda6630>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3dd4eba8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3dd00198>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3dd001d0>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3dccfc50>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3dc81208>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3dca9780>]],
               dtype=object)
```




```
In [15]: df3.hist(figsize=(40, 30))
```

```
Out[15]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3d902128>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3d934b70>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3d8cffd0>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3d8fe550>,
```

```
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3d8a5ac8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3d859080>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3d87f5f8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3d827ba8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3d827be0>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3d7e81d0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3d78f748>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f0a3d7b5cc0>]],
dtype=object)
```



2.2.3 Value counts for destination column

The column of Destination is the only non-numerical column. So, let us take a look at its values.

```
In [16]: # Print the value counts for specific columns
         col = "Destination"
         print('\nColumn Name:', col, df1[col].dtype)
         print(df1[col].value_counts())
```

```
Column Name: Destination object
Personal                                                        2521
```

PERSONAL	1375
personal	1265
CRS-Home-CRS	1169
HOME-CRS-HOME	805
CRS-HOME-CRS	571
Home-Office-Home	555
CRS-Home	533
crs-hme-crs	516
errands	493
HOME-CRS	466
Home-CRS	441
Office-Home-Office	435
crs-home-crs	426
Home-Office	406
CRS-HOME	373
hme-crs	322
CRS-LORESHO	260
crs-westlands-crs	248
posting errands	246
crs-hme	242
CRS-Westlands-CRS	236
Office - Home - Office	234
Office-Home	226
crs-bank-crs	224
CRS-Loresho	222
home-crs	220
Home-CRS-Home	189
crs-total-crs	183
crs-home	176
	...
CRS Official	1
FENESI-CRS-AGA KHAN	1
Crs-Westlands-Crs	1
crs-home-crs for 8th- 9th and 10th Jan	1
CRS-Holiday Inn-CRS	1
WEST POKOT TARTAR	1
crs to Nbi hospital to CRS	1
h/bay-ndhiwa	1
CR-RESIDENCE-CRS-CR-RESIDENCE	1
crs-Gisellas's res-crs	1
Crs - Res - Crs - Res	1
H/bay-town-ksm	1
CRS-Hangarian embassy	1
NAIROBI CRS-NISONYO-NAIROBI CRS	1
Home-CRS-Home-Sarit--Home	1
MERU-MARIMANTI-MERU	1
Crs-res-crs	1
isiolo-field	1

```
CRS - PM- RESIDENT - BACK CRS OFFICE.                          1
crs-Kenya leather-crs                                          1
CD of NakuruLantalinya primary                                 1
nbo-machkos-nbo                                                1
Diani-Mombasa                                                  1
CRS-HILLVIEW-LAVINGTON-CRS-LORESHO-CRS                         1
crs-msard-ind-town-crs                                         1
Mldi-Watamu                                                    1
Home - CRS - Home - Loresho - CRS - Home                       1
Home - CRS - Home - Loresho - Hill view - Home - CRS - Home    1
narok-naivashe-nbi                                             1
Migori Nairobi                                                 1
Name: Destination, Length: 23305, dtype: int64
```

In [17]:
```python
# Print the value counts for specific columns
col = "Destination"
print('\nColumn Name:', col, df2[col].dtype)
print(df2[col].value_counts())
```

```
Column Name: Destination object
CEV                                                             3331
EV                                                              2002
PERSO                                                            950
cev                                                              497
IVATO                                                            434
ev                                                               330
TANA                                                             228
CEV ABV                                                          187
FMR-CRS                                                          184
CRS                                                              174
CRS-FMR                                                          172
ANNICK-CRS                                                       166
PERSONAL                                                         158
FNR                                                              157
BMOI                                                             156
HOME OFFICE                                                      146
CRS- DOM- CRS                                                    131
EV FNR                                                           128
PERSONNEL                                                        123
CRS-MAISON-CRS                                                   116
FNR CEV                                                          115
DOM FMR                                                          115
CRS-ANNICK                                                       113
CEV TNR                                                          103
CRS-GALANA-CRS                                                   103
HOME OFFICE/OFFICE HOME                                           98
```

```
CRS - MAISON - CRS                                                 98
ANNICK- CRS                                                        98
CRS-HOME                                                           97
perso                                                              90
                                                                  ...
BUREAU ANDRAMANERA                                                  1
TMV-FEE- ANJAHABE-TMV                                               1
MANAKANA- MNJ                                                       1
BNI Bazarkely ODDIT                                                 1
CRS- BMOI- ANTSAHAVOLA- ANTANIMENA- CRS                             1
HTL- NEPTUNE- ODDIT- NEPTUNE- MARIMAR- HTL                          1
MIN COM USAID                                                       1
LAZAN'ANDROY (3 FOIS)-ABSRK-BUREAU-TANJONA-ONN-LEILAH-BUREAU        1
CRS-INFINITHE-CRS                                                   1
MASIABOAY - TANANTSOA                                               1
HTL VONDROZO.SAKELIRANO.HTL VONDROZO                                1
FNR-ABE-TNR                                                         1
LE DAUPHIN- TSIHOMBE                                                1
MNJ. AMBOHIMAHASOA SHELL                                            1
BRICKAVILLE-SAHAVALAINA-BRICKAVILLE GALANA                          1
AEROPORT BUREAU MADAUTO BUREAU                                      1
ARIVONIMAMO ILAFY                                                   1
TMV- BEFORONA -TMV                                                  1
MANAKANA NORD-MANAKARA                                              1
CRS - HOME                                                          1
MGH-SOFIA                                                           1
CRS- TSIMBAZAZA- CRS                                                1
VGN- VOHIMENA-VGN                                                   1
FARAFANGANA-TANGAINONY-MAROLAKA-FARAFANGANA                         1
ABV-TS/BE-F/D                                                       1
ANJOHY-FARAVOHITRA-AMPEFILOHA-ANALAKELY                             1
FNR APPRO-MNJR                                                      1
BEFA SUD- ANTANIMEVA- ANKILIKASY- ANTANIMEVA- MOROMBE               1
TAKE BEN END GABRIELLA FROM                                         1
CRS - ANDRAHARO - FUTURA - CRS                                      1
Name: Destination, Length: 28726, dtype: int64
```

In [18]:
```python
# Print the value counts for specific columns
col = "Destination"
print('\nColumn Name:', col, df3[col].dtype)
print(df3[col].value_counts())
```

```
Column Name: Destination object
COMMUTE                                     4532
PERSONAL                                    3946
SCHOOL RUN                                  3621
```

WUSE	1944
AUSTOMA	1415
FUELING	1147
SCHOOL RUNS	1000
WUSE 2	807
COMMUTTE	775
COMMUTEE	657
OFFICE TO AIRPORT	584
BANKING	582
OFFICE TO HOME	578
WUSE II	565
MAITAMA	553
HOTEL	548
BANK	544
HOME TO OFFICE	504
AIRPORT	493
SCHOOL	483
OFFICE TO JABI	473
OFFICE - HOME	466
JABI	462
FUELLING	379
PERSONAL USE	361
HOME	359
CENTRAL AREA	327
wuse	322
OFFICE	320
WUSE TO OFFICE	311
	...
JABI OOFICE	1
OFFICE TO GWARINA	1
LAFIA OFFICE TO NNPC	1
WFE MEETING	1
EMBASSY AND JABI	1
AGU TO OLO H.C	1
OLD GUEST HOUSE TOOFFICE	1
CORNOIL	1
OANDO FUNTUA TO ABUJA OFFICE	1
EKA	1
gaduwe	1
OFFICE - EGYPT EMBASSY	1
DROP LETTERS CENTRAL AREA	1
OFFICE TO RIDERS WORKSHOP	1
OFFICE =- HOME	1
MURIPHA -ECWA EGBE 18/10/2017	1
OFFICE BWARI SITE VISIT	1
OFFICE----SC BANK FEDEX OFFICE	1
OFFICE O BIZ CENTER	1
OFFICE TO VIO OFFICE AND ASOKORO	1

```
OFFICE TO OLD NEW HOUSE                          1
ABIA -ABA                                        1
OFF TO AREA 2                                    1
OFFICE - DUTSE TO PICK UP CAR AT CORRONET        1
GBOKO TO VANDEKYA                                1
V ANDTO GBOKO                                    1
GEROGE TOWN                                      1
OFFICE TO SAK FUEL STATION                       1
PERSONSL                                         1
HOTEL TO DOGERE                                  1
Name: Destination, Length: 26566, dtype: int64
```

3 Data Engineering

After reading and exploring the three data sets, I will do some data manipulations so that I can have a data set ready for machine learning algorithms to use.

3.1 Generate Sample Data Set

3.1.1 Take samples

Python Scikit learn library will run machine learning algorithms on the local in-memory data set. So, I will take 10,000 records from data sets so that it can take acceptable time to produce output.

```
In [19]: df1 = df1.sample(3000)
         df2 = df2.sample(3000)
         df3 = df3.sample(4000)
```

3.1.2 Combine three data sets

```
In [20]: df4 = pd.concat([df1, df2, df3])
         df4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 35836 to 28502
Data columns (total 13 columns):
ID             10000 non-null int64
Date           10000 non-null object
Country        10000 non-null int64
VehID          10000 non-null int64
Destination    9664 non-null object
KmInit         10000 non-null int64
KmFinal        10000 non-null int64
FuelBought     10000 non-null float64
AmountFuel     10000 non-null float64
CashFuel       10000 non-null float64
AmountCash     10000 non-null float64
```

```
FuelKm          10000 non-null int64
DriverID        10000 non-null int64
dtypes: float64(4), int64(7), object(2)
memory usage: 1.1+ MB
```

3.2 Create Columns based on Existing Ones

3.2.1 Calculate distance for every driving history record

```
In [21]: df4['Distance'] = abs(df4['KmFinal'] - df4['KmInit'])
```

3.2.2 Determine the directions of driving vehicle

```
In [22]: def usage_type(x):
             if isinstance(x, str):
                 x = x.lower()
                 result = None
                 if '-' in x:
                     wordList = x.strip().split('-')
                 elif (',') in x:
                     wordList = x.strip().split(',')
                 elif ('/') in x:
                     wordList = x.strip().split('/')
                 elif ('.') in x:
                     wordList = x.strip().split('.')
                 elif ('to') in x:
                     wordList = x.strip().split('to')
                 else:
                     wordList = x.strip().split()

                 first_word = wordList[0].strip()
                 last_word = wordList[len(wordList) - 1].strip()
                 #print(first_word, last_word)

                 if (first_word == last_word) and len(wordList) > 1:
                     result = 2
                 elif 'commute' in x:
                     result = 2
                 elif 'person' in x:
                     result = 1
                 else:
                     result = 1
                 return result
             else:
                 return 0

         df4['Directions'] = df4['Destination'].apply(usage_type)
```

3.2.3 Calculate how many stops during one driving history

```
In [23]: def get_stops(x):
             if isinstance(x, str):
                 x = x.lower()
                 result = None
                 if '-' in x:
                     wordList = x.strip().split('-')
                 elif (',') in x:
                     wordList = x.strip().split(',')
                 elif ('/') in x:
                     wordList = x.strip().split('/')
                 elif ('.') in x:
                     wordList = x.strip().split('.')
                 elif ('to') in x:
                     wordList = x.strip().split('to')
                 else:
                     wordList = x.strip().split()

                 result = len(set(wordList))
                 return result
             else:
                 return 0

         df4['Stops'] = df4['Destination'].apply(get_stops)
```

3.2.4 Calculate how fuel was added and purchased

```
In [24]: df4['When_Fuel_Added'] = df4.apply(lambda x: x['FuelKm'] - x['KmInit'] if (x['FuelKm']
         df4['is_Fuel_Added'] = df4.apply(lambda x: 1 if (x['AmountFuel'] > 0) or (x['AmountCas
         df4['is_Cash_Paid'] = df4.apply(lambda x: 1 if x['AmountCash'] > 0 else 0, axis=1)
```

3.3 Extract Data Set for Machine Learning Use

3.3.1 Query data set

```
In [25]: df = df4.loc[(df4['Distance'] >0), ['Country', 'Directions', 'Distance', 'Stops', 'is_
         df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 35836 to 28502
Data columns (total 7 columns):
Country            10000 non-null int64
Directions         10000 non-null int64
Distance           10000 non-null int64
Stops              10000 non-null int64
is_Fuel_Added      10000 non-null int64
is_Cash_Paid       10000 non-null int64
When_Fuel_Added    10000 non-null int64
```

```
dtypes: int64(7)
memory usage: 625.0 KB
```

3.3.2 Data Set Profiling

```
In [26]: pandas_profiling.ProfileReport(df)
```

```
/usr/local/lib/python3.6/dist-packages/matplotlib/font_manager.py:1241: UserWarning: findfont:
  (prop.get_family(), self.defaultFamily[fontext]))
/usr/local/lib/python3.6/dist-packages/matplotlib/font_manager.py:1241: UserWarning: findfont:
  (prop.get_family(), self.defaultFamily[fontext]))
/usr/local/lib/python3.6/dist-packages/matplotlib/font_manager.py:1241: UserWarning: findfont:
  (prop.get_family(), self.defaultFamily[fontext]))
/usr/local/lib/python3.6/dist-packages/matplotlib/font_manager.py:1241: UserWarning: findfont:
  (prop.get_family(), self.defaultFamily[fontext]))
/usr/local/lib/python3.6/dist-packages/matplotlib/font_manager.py:1241: UserWarning: findfont:
  (prop.get_family(), self.defaultFamily[fontext]))
```

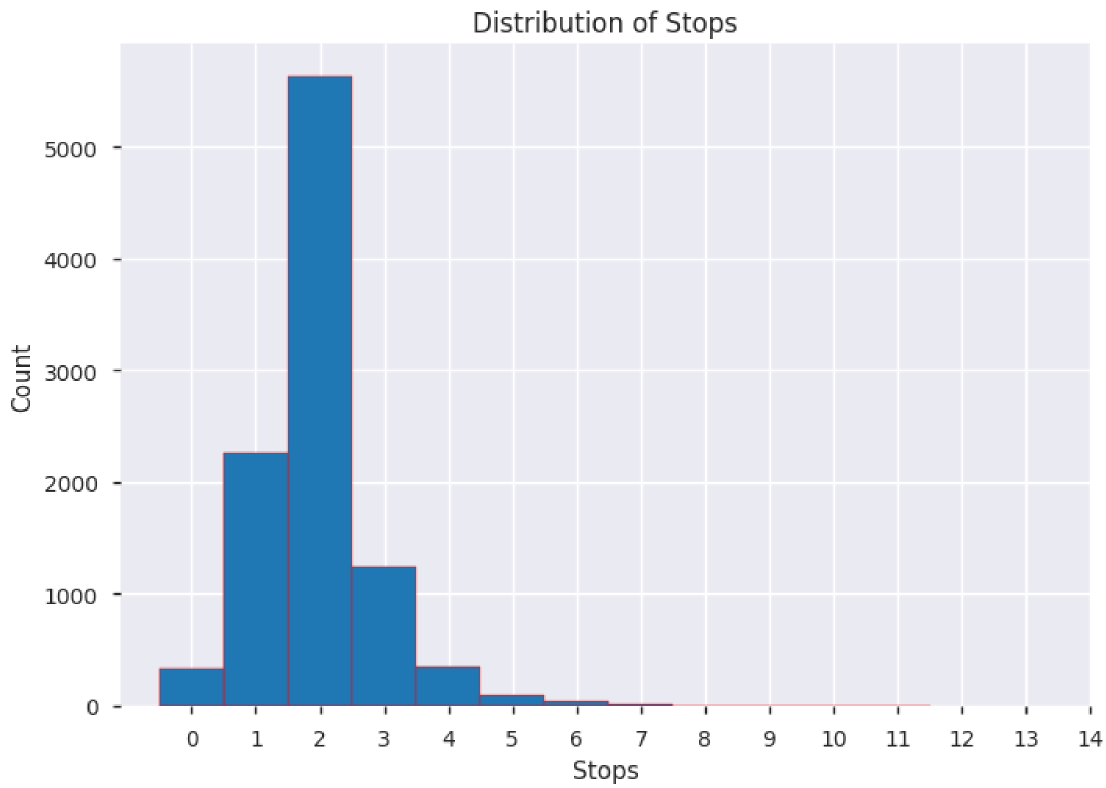
```
Out[26]: <pandas_profiling.ProfileReport at 0x7f0a3f93b518>
```

3.4 Data Exploration for Trending

3.4.1 Column distributions

```
In [27]: # Bar plot of Stops
         plt.bar(df['Stops'].value_counts().index, df['Stops'].value_counts().values,
                 fill = 'blue', edgecolor = 'r', width = 1)
         plt.xlabel('Stops')
         plt.ylabel('Count')
         plt.title('Distribution of Stops')
         plt.xticks(list(range(0, 15)))
```

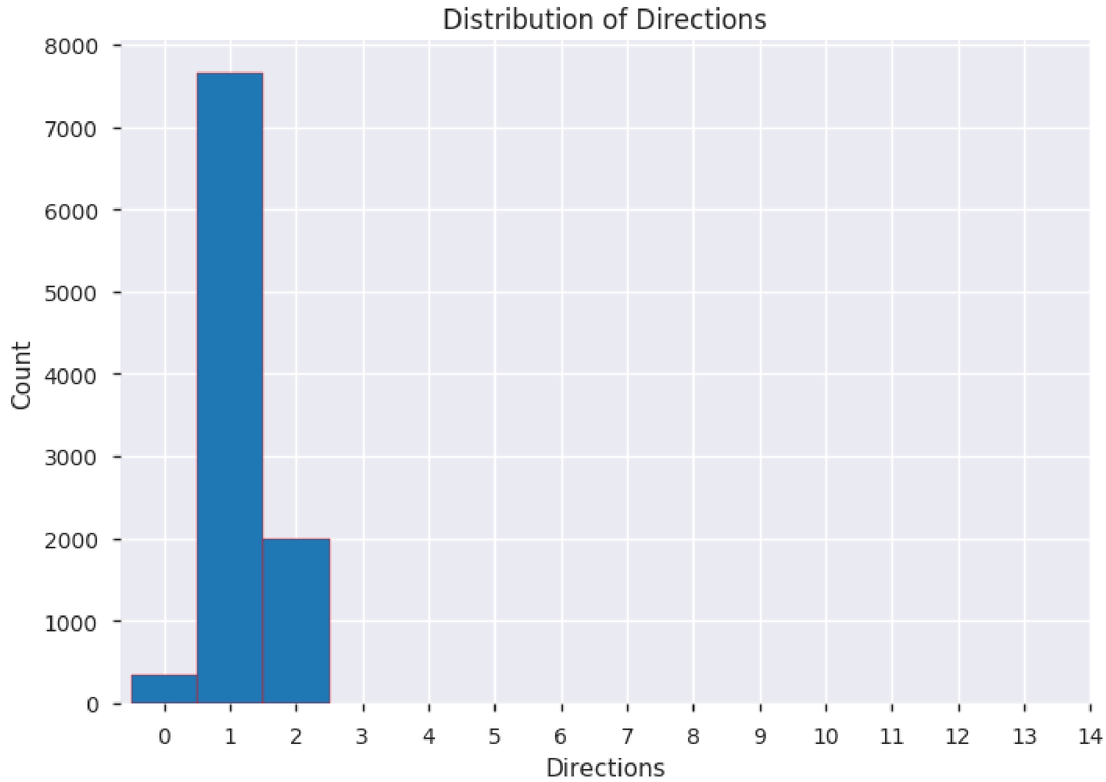
```
Out[27]: ([<matplotlib.axis.XTick at 0x7f0a39b46438>,
           <matplotlib.axis.XTick at 0x7f0a39b42c88>,
           <matplotlib.axis.XTick at 0x7f0a39b429b0>,
           <matplotlib.axis.XTick at 0x7f0a39af8eb8>,
           <matplotlib.axis.XTick at 0x7f0a39ae5358>,
           <matplotlib.axis.XTick at 0x7f0a39b03748>,
           <matplotlib.axis.XTick at 0x7f0a39b03c88>,
           <matplotlib.axis.XTick at 0x7f0a39b0d208>,
           <matplotlib.axis.XTick at 0x7f0a39b0d748>,
           <matplotlib.axis.XTick at 0x7f0a39b0dc88>,
           <matplotlib.axis.XTick at 0x7f0a39a94208>,
           <matplotlib.axis.XTick at 0x7f0a39a94748>,
           <matplotlib.axis.XTick at 0x7f0a39b0d668>,
           <matplotlib.axis.XTick at 0x7f0a39b03668>,
           <matplotlib.axis.XTick at 0x7f0a39a94e10>],
          <a list of 15 Text xticklabel objects>)
```




```
In [28]: # Bar plot of Directions
         plt.bar(df['Directions'].value_counts().index, df['Directions'].value_counts().values
                 fill = 'blue', edgecolor = 'r', width = 1)
         plt.xlabel('Directions')
         plt.ylabel('Count')
         plt.title('Distribution of Directions')
         plt.xticks(list(range(0, 15)))
```

```
Out[28]: ([<matplotlib.axis.XTick at 0x7f0a39a1ab70>,
           <matplotlib.axis.XTick at 0x7f0a39a1a400>,
           <matplotlib.axis.XTick at 0x7f0a39a1a2b0>,
           <matplotlib.axis.XTick at 0x7f0a39a3a7f0>,
           <matplotlib.axis.XTick at 0x7f0a39a3ac88>,
           <matplotlib.axis.XTick at 0x7f0a39a44208>,
           <matplotlib.axis.XTick at 0x7f0a39a44748>,
           <matplotlib.axis.XTick at 0x7f0a39a44c88>,
           <matplotlib.axis.XTick at 0x7f0a39a4b208>,
           <matplotlib.axis.XTick at 0x7f0a39a4b748>,
           <matplotlib.axis.XTick at 0x7f0a39a446d8>,
           <matplotlib.axis.XTick at 0x7f0a39a3a780>,
           <matplotlib.axis.XTick at 0x7f0a39a4b550>,
           <matplotlib.axis.XTick at 0x7f0a399d3358>,
```

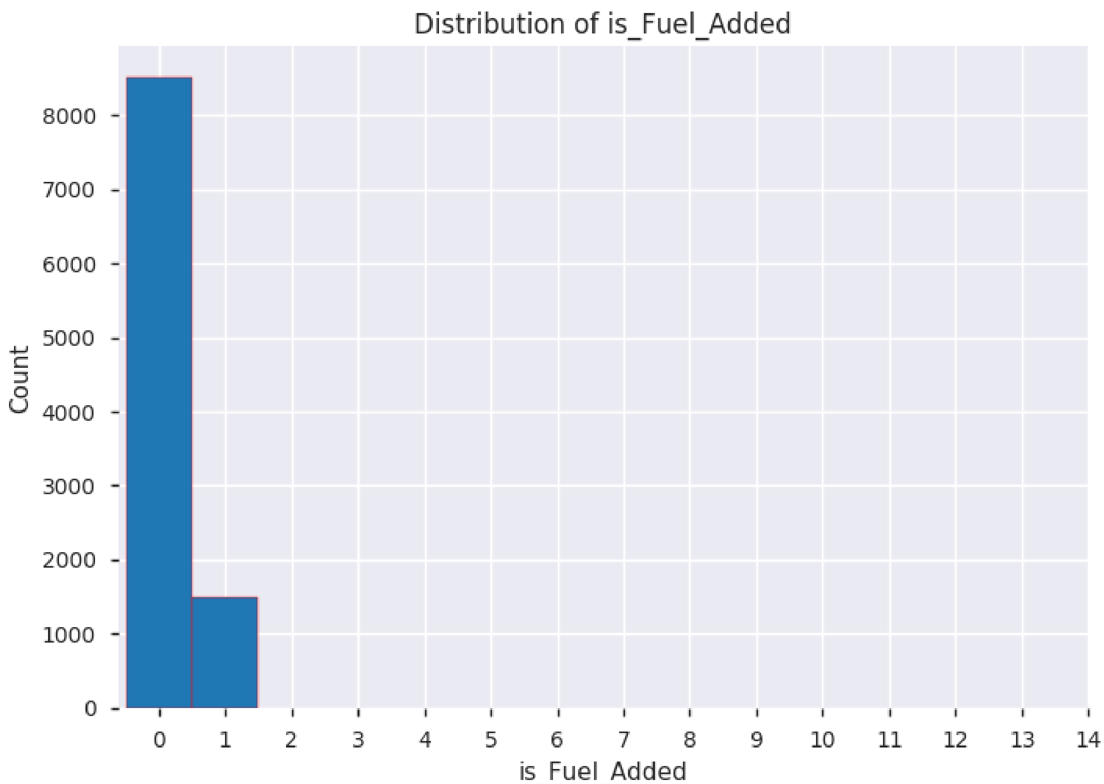
```
 <matplotlib.axis.XTick at 0x7f0a399d3898>],
<a list of 15 Text xticklabel objects>)
```




```
In [29]: # Bar plot of is_Fuel_Added
         plt.bar(df['is_Fuel_Added'].value_counts().index, df['is_Fuel_Added'].value_counts().v
                 fill = 'blue', edgecolor = 'r', width = 1)
         plt.xlabel('is_Fuel_Added')
         plt.ylabel('Count')
         plt.title('Distribution of is_Fuel_Added')
         plt.xticks(list(range(0, 15)))
```

```
Out[29]: ([<matplotlib.axis.XTick at 0x7f0a3999e5c0>,
           <matplotlib.axis.XTick at 0x7f0a39993e10>,
           <matplotlib.axis.XTick at 0x7f0a39993b38>,
           <matplotlib.axis.XTick at 0x7f0a399b9ef0>,
           <matplotlib.axis.XTick at 0x7f0a399c1400>,
           <matplotlib.axis.XTick at 0x7f0a399c1940>,
           <matplotlib.axis.XTick at 0x7f0a399c1e80>,
           <matplotlib.axis.XTick at 0x7f0a399ca400>,
           <matplotlib.axis.XTick at 0x7f0a399ca940>,
           <matplotlib.axis.XTick at 0x7f0a399cae80>,
           <matplotlib.axis.XTick at 0x7f0a39951400>,
```

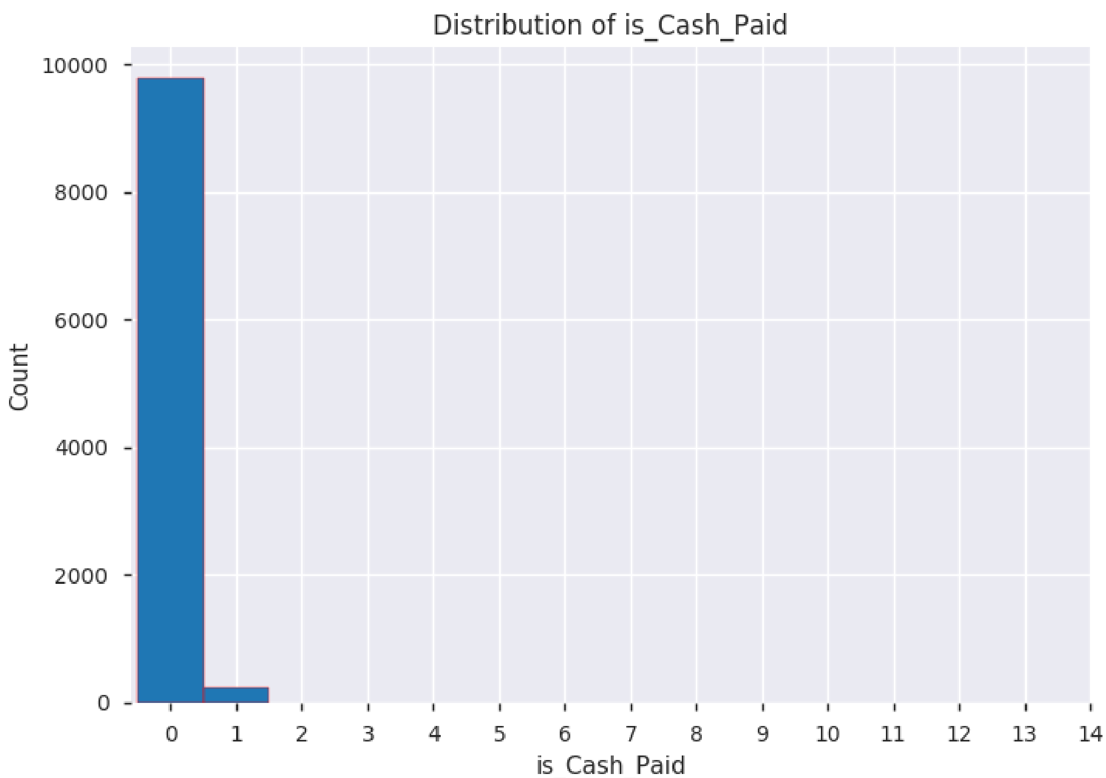
```
 <matplotlib.axis.XTick at 0x7f0a399ca8d0>,
 <matplotlib.axis.XTick at 0x7f0a399b9fd0>,
 <matplotlib.axis.XTick at 0x7f0a39951940>,
 <matplotlib.axis.XTick at 0x7f0a39951e80>],
<a list of 15 Text xticklabel objects>)
```



In [30]:
```
# Bar plot of is_Cash_Paid
plt.bar(df['is_Cash_Paid'].value_counts().index, df['is_Cash_Paid'].value_counts().va
        fill = 'blue', edgecolor = 'r', width = 1)
plt.xlabel('is_Cash_Paid')
plt.ylabel('Count')
plt.title('Distribution of is_Cash_Paid')
plt.xticks(list(range(0, 15)))
```

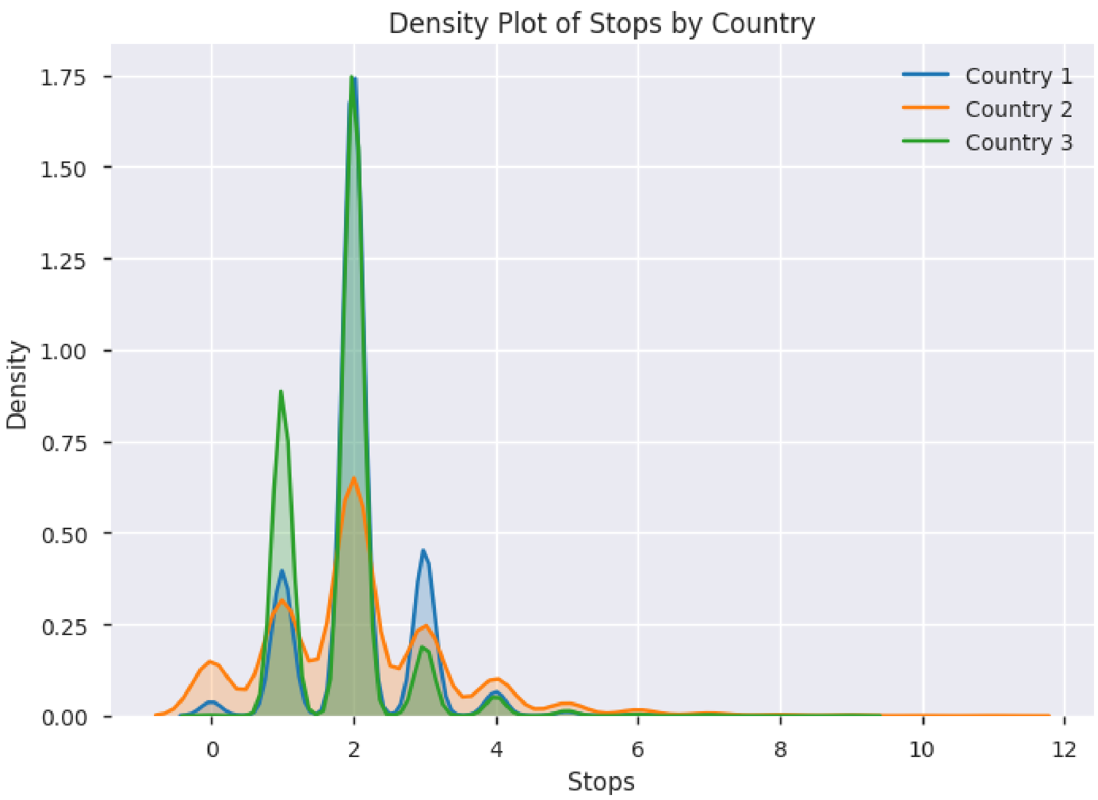
Out[30]:
```
([<matplotlib.axis.XTick at 0x7f0a39918cf8>,
  <matplotlib.axis.XTick at 0x7f0a39918588>,
  <matplotlib.axis.XTick at 0x7f0a399182b0>,
  <matplotlib.axis.XTick at 0x7f0a3993d6a0>,
  <matplotlib.axis.XTick at 0x7f0a3993db38>,
  <matplotlib.axis.XTick at 0x7f0a3993d978>,
  <matplotlib.axis.XTick at 0x7f0a399475f8>,
  <matplotlib.axis.XTick at 0x7f0a39947b38>,
```

```
 <matplotlib.axis.XTick at 0x7f0a39947978>,
 <matplotlib.axis.XTick at 0x7f0a3994c5f8>,
 <matplotlib.axis.XTick at 0x7f0a3994cb38>,
 <matplotlib.axis.XTick at 0x7f0a399476a0>,
 <matplotlib.axis.XTick at 0x7f0a39947c88>,
 <matplotlib.axis.XTick at 0x7f0a3994ce48>,
 <matplotlib.axis.XTick at 0x7f0a398d65f8>],
<a list of 15 Text xticklabel objects>)
```

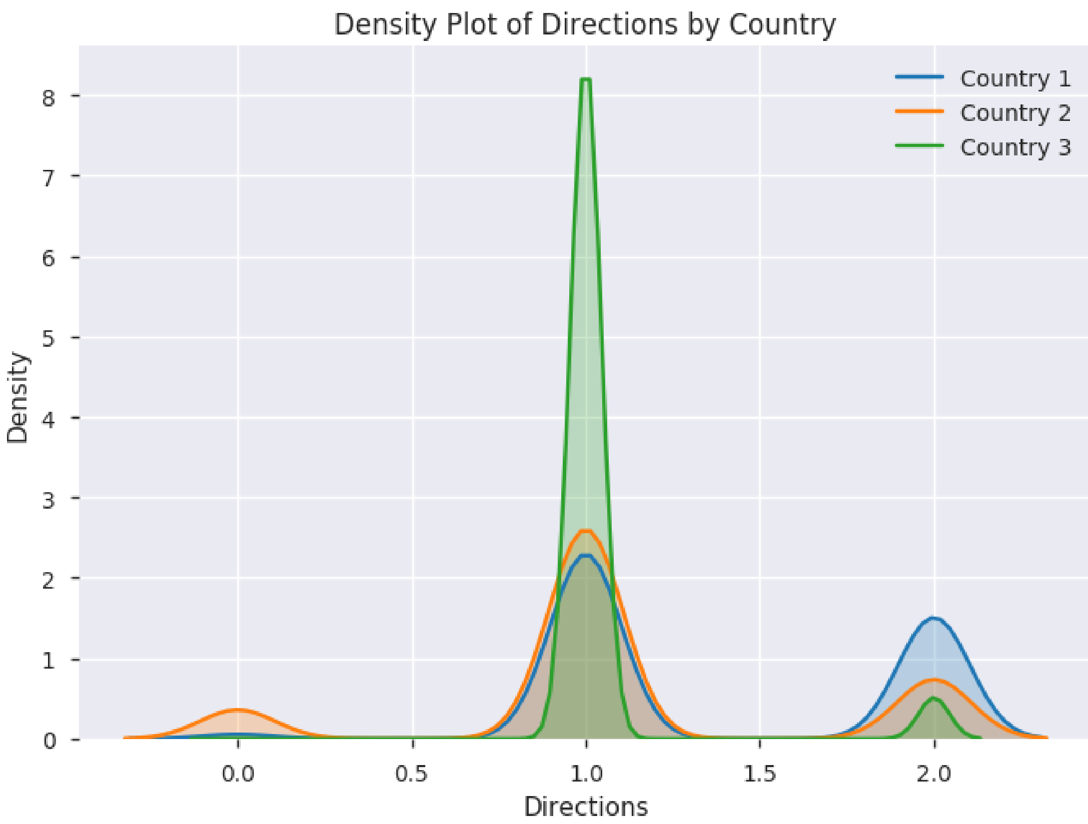


3.4.2 Column distributions by different countries

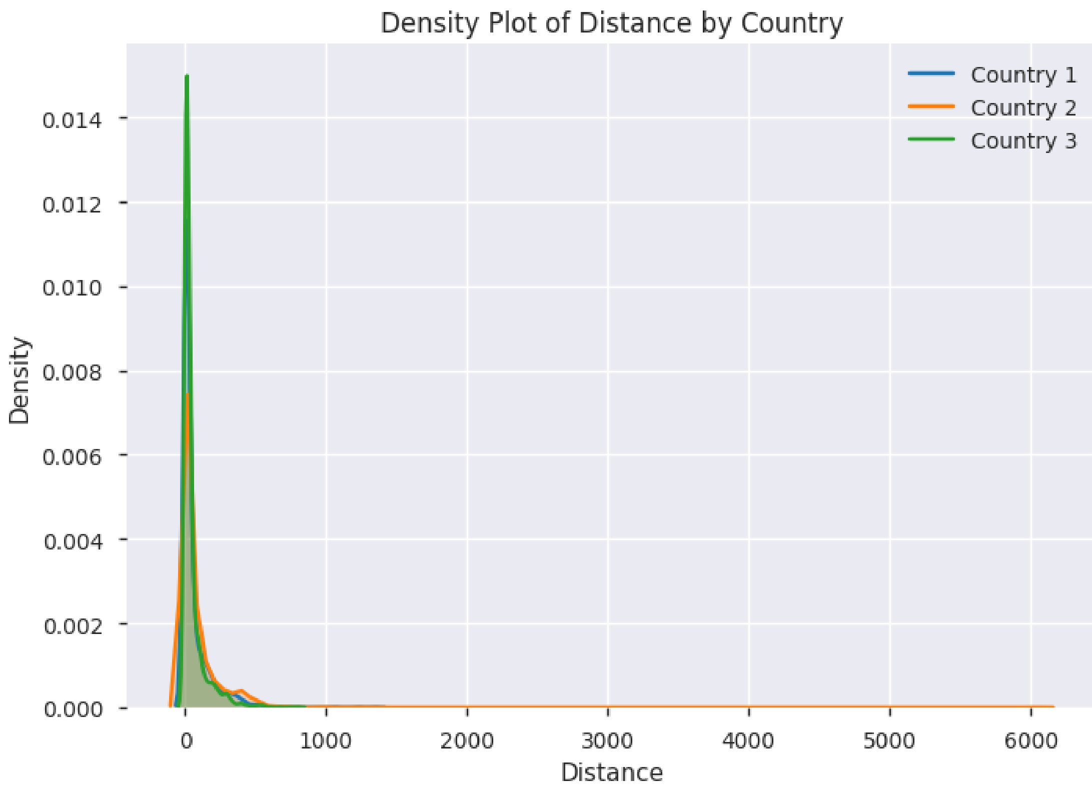
In [31]:
```
# Stops distribution by Country
sns.kdeplot(df.loc[df['Country'] == 1, 'Stops'], label = 'Country 1', shade = True)
sns.kdeplot(df.loc[df['Country'] == 2, 'Stops'], label = 'Country 2', shade = True)
sns.kdeplot(df.loc[df['Country'] == 3, 'Stops'], label = 'Country 3', shade = True)
plt.xlabel('Stops')
plt.ylabel('Density')
plt.title('Density Plot of Stops by Country')
```



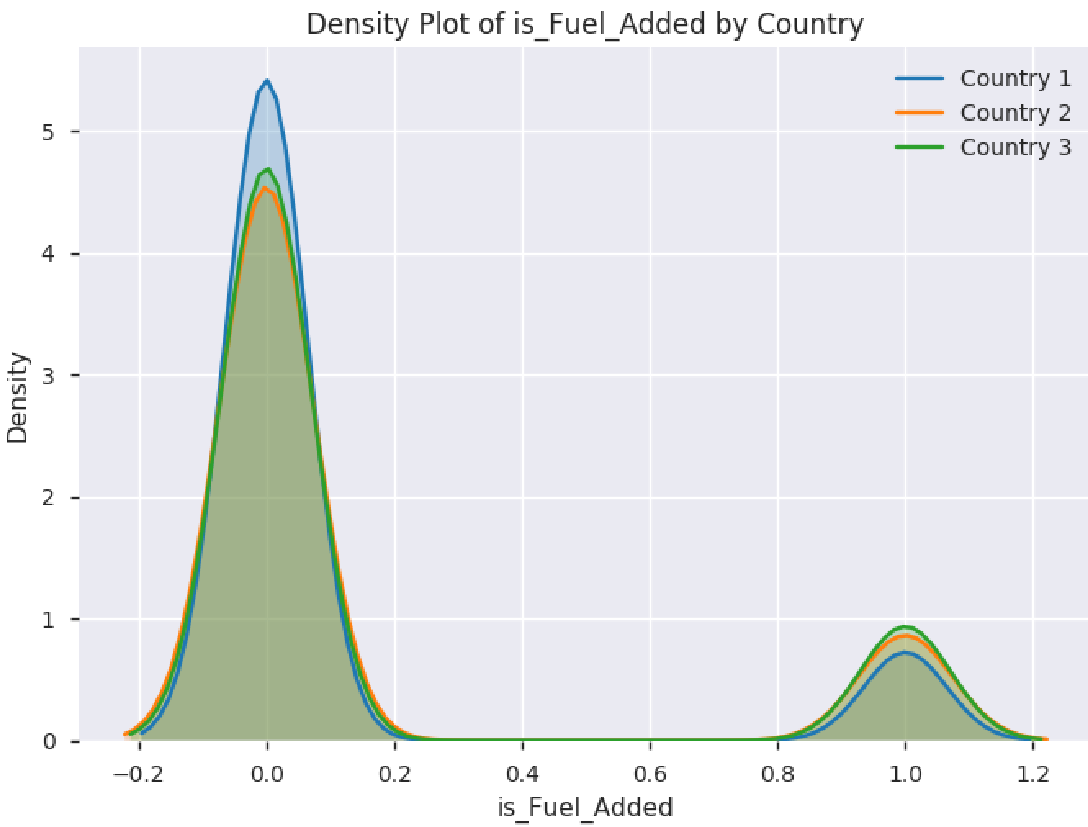

```
In [32]: # Directions distribution by Country
         sns.kdeplot(df.loc[df['Country'] == 1, 'Directions'], label = 'Country 1', shade = Tru
         sns.kdeplot(df.loc[df['Country'] == 2, 'Directions'], label = 'Country 2', shade = Tru
         sns.kdeplot(df.loc[df['Country'] == 3, 'Directions'], label = 'Country 3', shade = Tru
         plt.xlabel('Directions')
         plt.ylabel('Density')
         plt.title('Density Plot of Directions by Country')
```



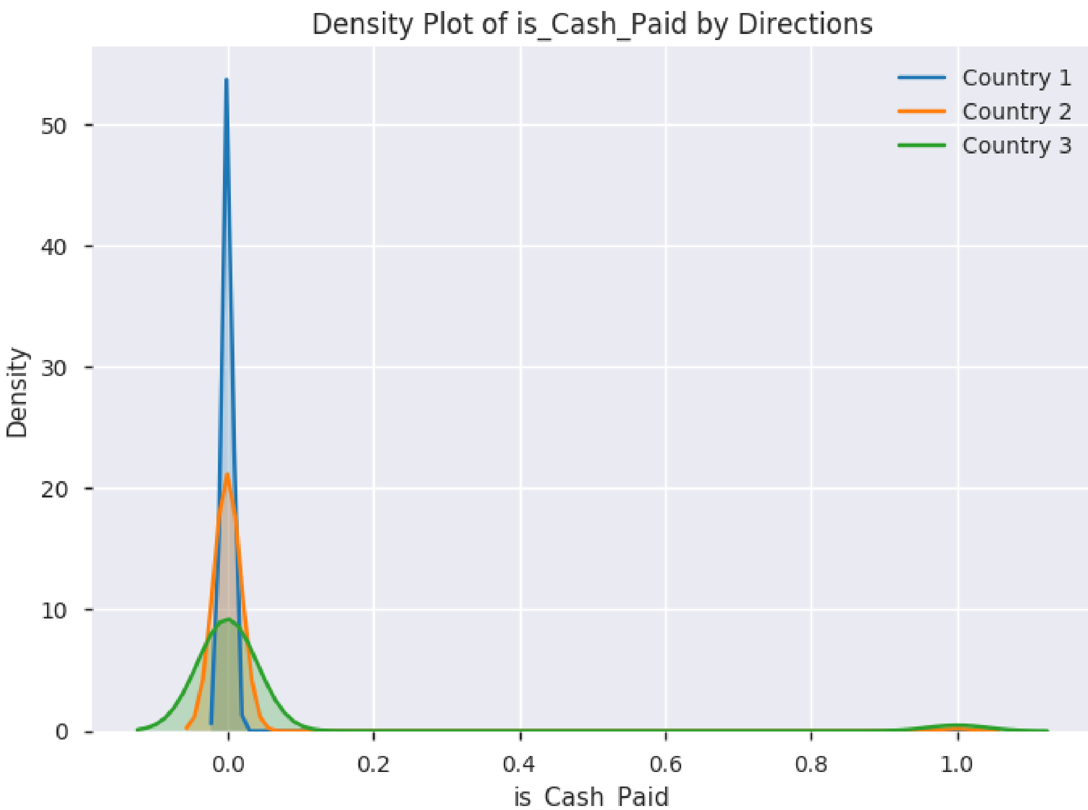

```
In [33]: # Distance distribution by Country
         sns.kdeplot(df.loc[df['Country'] == 1, 'Distance'], label = 'Country 1', shade = True
         sns.kdeplot(df.loc[df['Country'] == 2, 'Distance'], label = 'Country 2', shade = True
         sns.kdeplot(df.loc[df['Country'] == 3, 'Distance'], label = 'Country 3', shade = True
         plt.xlabel('Distance')
         plt.ylabel('Density')
         plt.title('Density Plot of Distance by Country')
```




```
In [34]: # is_Fuel_Added distribution by Country
         sns.kdeplot(df.loc[df['Country'] == 1, 'is_Fuel_Added'], label = 'Country 1', shade =
         sns.kdeplot(df.loc[df['Country'] == 2, 'is_Fuel_Added'], label = 'Country 2', shade =
         sns.kdeplot(df.loc[df['Country'] == 3, 'is_Fuel_Added'], label = 'Country 3', shade =
         plt.xlabel('is_Fuel_Added')
         plt.ylabel('Density')
         plt.title('Density Plot of is_Fuel_Added by Country')
```




```
In [35]: # is_Cash_Paid distribution by Country
         sns.kdeplot(df.loc[df['Country'] == 1, 'is_Cash_Paid'], label = 'Country 1', shade = T
         sns.kdeplot(df.loc[df['Country'] == 2, 'is_Cash_Paid'], label = 'Country 2', shade = T
         sns.kdeplot(df.loc[df['Country'] == 3, 'is_Cash_Paid'], label = 'Country 3', shade = T
         plt.xlabel('is_Cash_Paid')
         plt.ylabel('Density')
         plt.title('Density Plot of is_Cash_Paid by Directions')
```



3.4.3 Numerical correlations

```
In [36]: # Correlations of numerical values
         df.corr()['Country'].sort_values()

Out[36]: Directions        -0.284280
         Stops             -0.109562
         Distance          -0.043645
         When_Fuel_Added   -0.004904
         is_Fuel_Added      0.054476
         is_Cash_Paid       0.137214
         Country            1.000000
         Name: Country, dtype: float64
```

3.5 Generate Training and Testing Data Sets

3.5.1 Select most correlated 3 columns

```
In [37]: # Find the most correlated columns with the country and returns training and testing
         def ML_DataSet(df):
             # Targets are countries
             labels = df['Country']
```

```
    # df = pd.get_dummies(df)
    # Find correlations with the country
    most_correlated = df.corr().abs()['Country'].sort_values(ascending=False)

    # Maintain the top 3 most correlation columns with country
    most_correlated = most_correlated[:4]
    df = df.loc[:, most_correlated.index]

    # Split into training/testing sets with 25% split
    X_train, X_test, y_train, y_test = train_test_split(df, labels,
                                                        test_size = 0.25,
                                                        random_state=42)

    return X_train, X_test, y_train, y_test
```

In [38]:
```
X_train, X_test, y_train, y_test = ML_DataSet(df)
X_train.head()
```

Out[38]:

	Country	Directions	is_Cash_Paid	Stops
29384	2	1	0	2
14827	2	1	0	1
2548	3	1	0	1
43820	3	1	0	2
49827	1	2	0	3

In [39]:
```
X_train.head()
```

Out[39]:

	Country	Directions	is_Cash_Paid	Stops
29384	2	1	0	2
14827	2	1	0	1
2548	3	1	0	1
43820	3	1	0	2
49827	1	2	0	3

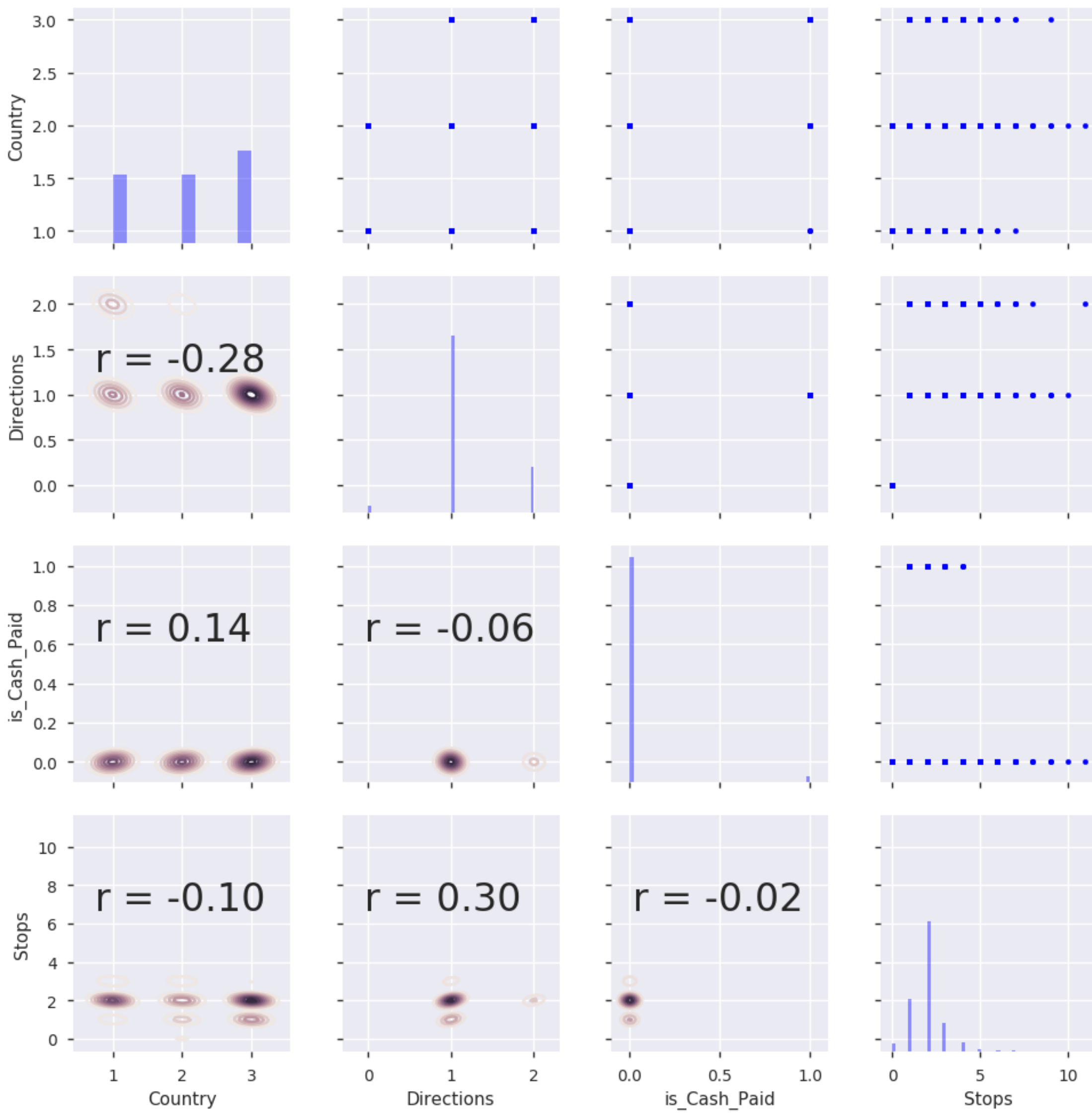
In [40]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(7500, 4)
(2500, 4)
```

3.5.2 Plots of Selected Columns Correlation Coefficient

In [41]:
```
# Calculate correlation coefficient
def COEFF(x, y, **kws):
    r, tmp = stats.pearsonr(x, y)
    p = plt.gca()
    p.annotate("r = {:.2f}".format(r), xy=(.1, .6), xycoords=p.transAxes, size = 24)
```

```python
# Visualize the results
cmap = sns.cubehelix_palette(light=1, dark=0.1, hue=0.5, as_cmap=True)
sns.set_context(font_scale=2)
g = sns.PairGrid(X_train)
g.map_upper(plt.scatter, s=10, color = 'blue')
g.map_diag(sns.distplot, kde=False, color = 'blue')
g.map_lower(sns.kdeplot, cmap = cmap)
g.map_lower(COEFF);
```



4 Predictive Analytics with Machine Learning Approaches

4.1 Setup Metrics

For this data analytics project, I will use two standard metrics (wiki definitions) from :

- Mean Absolute Error (MAE): is an interpretable & scale-dependent accuracy measure of difference between two continuous variables and is average vertical distance between each point and the identity line.
- Root Mean Squared Error (RMSE): is a frequently used & scale-dependent measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed. It represents the square root of the second sample moment of the differences between predicted values and observed values or the quadratic mean of these differences. RMSE is always non-negative, and a value of 0 (almost never achieved in practice) would indicate a perfect fit to the data. In general, a lower RMSE is better than a higher one.

For more information and discussions around those two metrices, here is a discussion.

```
In [42]: # Function to calculate mae and rmse
         def evaluate_predictions(predictions, real):
             mae = np.mean(abs(predictions - real))
             rmse = np.sqrt(np.mean((predictions - real) ** 2))

             return mae, rmse
```

4.2 Setup Baseline

For a regression machine learning approach, a simple baseline is to guess the median value on the training set for all testing cases. I will evaluate if machine learning approach can be better than the simple baseline.

```
In [43]: # Baseline is the median
         baseline_pred = X_train['Country'].median()
         baseline_preds = [baseline_pred for _ in range(len(X_test))]
         real = X_test['Country']

In [44]: mb_mae, mb_rmse = evaluate_predictions(baseline_preds, real)
         print('Baseline  MAE: {:.4f}'.format(mb_mae))
         print('Baseline RMSE: {:.4f}'.format(mb_rmse))

Baseline  MAE: 0.6948
Baseline RMSE: 0.8335
```

4.3 Machine Learning Approaches

```
In [45]: # Evaluate machine learning approaches by training on training set and testing on tes
         def evaluate(X_train, X_test, y_train, y_test):
             # ML approach list including baseline above
```

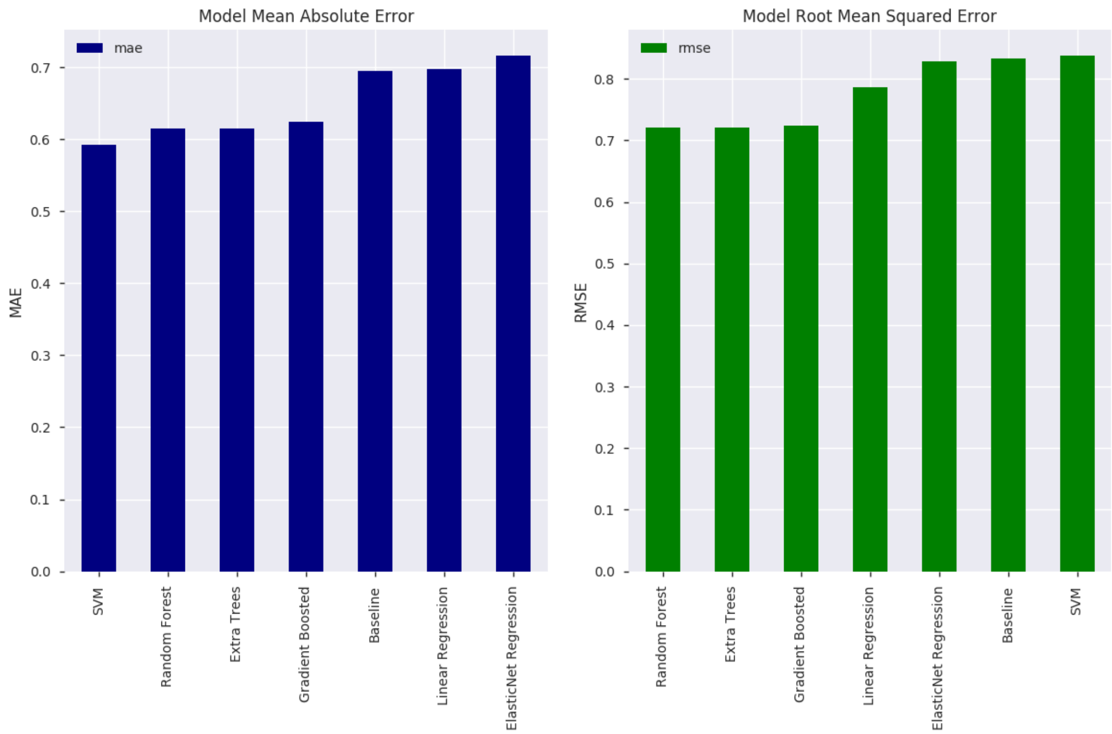
```
    model_name_list = ['Linear Regression', 'ElasticNet Regression', 'Random Forest',
                        'Extra Trees', 'SVM', 'Gradient Boosted', 'Baseline']
    X_train = X_train.drop(columns='Country')
    X_test = X_test.drop(columns='Country')

    # Call ML approaches
    model1 = LinearRegression()
    model2 = ElasticNet(alpha=1.0, l1_ratio=0.5)
    model3 = RandomForestRegressor(n_estimators=50)
    model4 = ExtraTreesRegressor(n_estimators=50)
    model5 = SVR(kernel='rbf', degree=3, C=1.0, gamma='auto')
    model6 = GradientBoostingRegressor(n_estimators=20)

    # Create a dataframe for results
    results = pd.DataFrame(columns=['mae', 'rmse'], index = model_name_list)

    # Train and predict with each model
    for i, model in enumerate([model1, model2, model3, model4, model5, model6]):
        model.fit(X_train, y_train)
        predictions = model.predict(X_test)
        # calculate metrics
        mae = np.mean(abs(predictions - y_test))
        rmse = np.sqrt(np.mean((predictions - y_test) ** 2))
        # put results into the dataframe
        model_name = model_name_list[i]
        results.loc[model_name, :] = [mae, rmse]

    # calculate baseline
    baseline = np.median(y_train)
    baseline_mae = np.mean(abs(baseline - y_test))
    baseline_rmse = np.sqrt(np.mean((baseline - y_test) ** 2))

    # fill dataframe for results
    results.loc['Baseline', :] = [baseline_mae, baseline_rmse]

    return results
```

In [46]:
```
# run ML approach and evaluation
results = evaluate(X_train, X_test, y_train, y_test)
```

4.4 Visual Comparison of ML Approaches

In [47]:
```
figsize(12, 8)
matplotlib.rcParams['font.size'] = 16

# MAE plot
ax =  plt.subplot(1, 2, 1)
results.sort_values('mae', ascending = True).plot.bar(y = 'mae', color = 'navy', ax =
```

```
plt.title('Model Mean Absolute Error')
plt.ylabel('MAE')

# RMSE plot
ax = plt.subplot(1, 2, 2)
results.sort_values('rmse', ascending = True).plot.bar(y = 'rmse', color = 'g', ax = a
plt.title('Model Root Mean Squared Error')
plt.ylabel('RMSE')
plt.tight_layout()
```



In [48]:
```
# show quantitative results
results
```

Out[48]:

	mae	rmse
Linear Regression	0.698025	0.786011
ElasticNet Regression	0.715794	0.827868
Random Forest	0.614889	0.721275
Extra Trees	0.615302	0.721358
SVM	0.591881	0.837453
Gradient Boosted	0.624126	0.724332
Baseline	0.6948	0.833547

In [49]:
```
print('The Random Forest is {:0.2f}% better on MAE than the baseline.'.format(
    (100 * abs(results.loc['Random Forest', 'mae'] - results.loc['Baseline', 'mae']))
```

```
    print('The Random Forest is {:0.2f}% better on RMSE than the baseline.'.format(
        (100 * abs(results.loc['Random Forest', 'rmse'] - results.loc['Baseline', 'rmse'])
```

```
The Random Forest is 11.50% better on MAE than the baseline.
The Random Forest is 13.47% better on RMSE than the baseline.
```

4.5 Interpretable Formula with Ordinary Linear Regression

```
In [50]: lr = LinearRegression()
         lr.fit(X_train.drop(columns='Country'), y_train)

         formula = 'Country = %0.2f +' % lr.intercept_
         for i, col in enumerate(X_train.columns[1:]):
             formula += ' %0.2f * %s +' % (lr.coef_[i], col)

         print(' '.join(formula.split(' ')[:-1]))
```

```
Country = 2.70 + -0.50 * Directions + 0.66 * is_Cash_Paid + -0.02 * Stops
```

5 Conclusions

In this notebook I went through major steps to demonstrate how a data analytic project can be implemented. At the end, several machine learning approaches were evaluated and compared based on two standard metrics, such as MAE and RMSE.

During this project implementation, one of limitations is the performance of data manipulation and machine learning training processes. When combining the whole three data sets as a training set, it takes a long time (more than half day) to manipulate and train the data. This is a known issue with pandas and scikit-learn libraries running on a local machine. That is, the running time can be an exponential growth on the size of training data set. That is why I only take 10,000 records from the whole three data sets as the training data set. During this project implementation, one of limitations is the performance of data manipulation and machine learning training processes. When combining the whole three data sets as a training set, it takes a long time (more than half day) to manipulate and train the data. This is a known issue with pandas and scikit-learn libraries running on a local machine. That is, the running time can be an exponential growth on the size of training data set. That is why I only take 10,000 records from the whole three data sets as the training data set.

A scalable solution for this problem is to use Spark (with pyspark library) on a Hadoop cluster (a group of multiple servers). In another notebook, I will demonstrate this solution for the same VMS data sets.