Data_Analytics_Project_Spark

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 **pyspark.sql** & **Optimus** to run data processing on Spark, **Optimus** to profile the data, **matplotlib** & **seaborn** to visualize the data and machine learning results, **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 [2]:
```
import findspark
findspark.init('/usr/local/spark/spark-2.4.0-bin-hadoop2.7')

import pyspark
# Load pyspark session and others
from pyspark import SQLContext
from pyspark.sql.session import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, BooleanType, Intege
from pyspark.sql import Row
from pyspark.sql.functions import udf, col, expr, when

# Create Optimus Context
from optimus import Optimus
op = Optimus(master="local", app_name="Data_Analytics", verbose=True)
# Create Spark SQL Context
sqlContext = SQLContext(op.sc)
```

In [73]:
```
# 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]:
```
# Pandas and numpy for converting from Spark dataframe into Pandas dataframe
import pandas as pd
import numpy as np
# Make the random numbers predictable
np.random.seed(42)
```

In [5]:
```
# Scipy for stats analysis
import scipy
from scipy import stats
```

In [6]:
```
# 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 [7]:
```
# Read three data sets
df1 = op.load.csv("../datasets/1_Log.csv",header=True)
df2 = op.load.csv("../datasets/2_Log.csv",header=True)
df3 = op.load.csv("../datasets/3_Log.csv",header=True)
```

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 [8]: df1.table(10)
        # df1.show(10)
```

```
<IPython.core.display.HTML object>
```

```
In [9]: df2.table(10)
        # df2.show(10)
```

```
<IPython.core.display.HTML object>
```

```
In [10]: df3.table(10)
         # df3.show(10)
```

```
<IPython.core.display.HTML object>
```

```
In [11]: #use SQL script to do the jobs
         sample_df = df1.cache()
         sample_df.registerTempTable('sample_table')
         sqlContext.sql('select * from sample_table order by ID desc').table(10)
```

```
<IPython.core.display.HTML object>
```

2.2 Data Set Information

```
In [14]: print("First data set - df1: (", df1.count(), ",", df1.cols.count(), ")")
         print("Second data set - df2: (", df2.count(), ",", df2.cols.count(), ")")
         print("Third data set - df3: (", df3.count(), ",", df3.cols.count(), ")")
         print("-----------------------------------")
         df1.dtypes
         print("-----------------------------------")
         df2.dtypes
         print("-----------------------------------")
         df3.dtypes
```

```
First data set - df1: ( 64497 , 13 )
Second data set - df2: ( 61200 , 13 )
Third data set - df3: ( 93447 , 13 )
-----------------------------------
-----------------------------------
-----------------------------------
```

Out[14]:
```
[('ID', 'int'),
 ('Date', 'string'),
 ('Country', 'int'),
 ('VehID', 'int'),
 ('Destination', 'string'),
 ('KmInit', 'int'),
 ('KmFinal', 'int'),
 ('FuelBought', 'double'),
 ('AmountFuel', 'double'),
 ('CashFuel', 'double'),
 ('AmountCash', 'double'),
 ('FuelKm', 'int'),
 ('DriverID', 'int')]
```

2.2.1 Describe for numeric columns

In [15]:
```
df1.describe().table()
# df1.describe().toPandas().transpose()
```

```
<IPython.core.display.HTML object>
```

In [16]:
```
df2.describe().table()
```

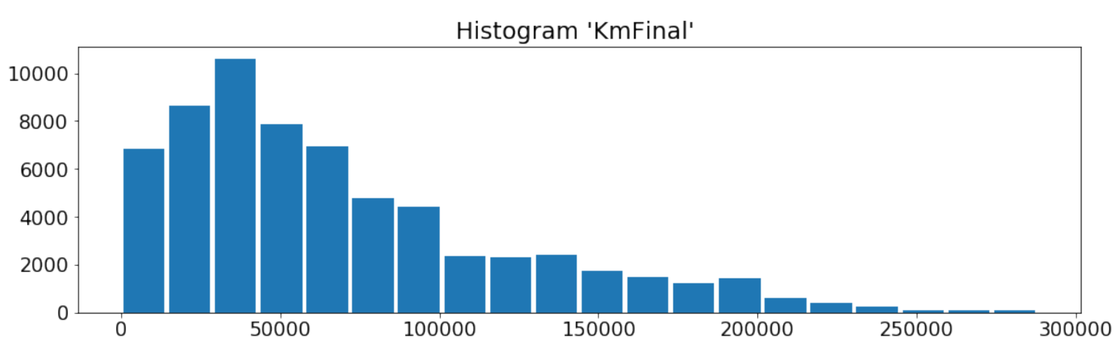
```
<IPython.core.display.HTML object>
```

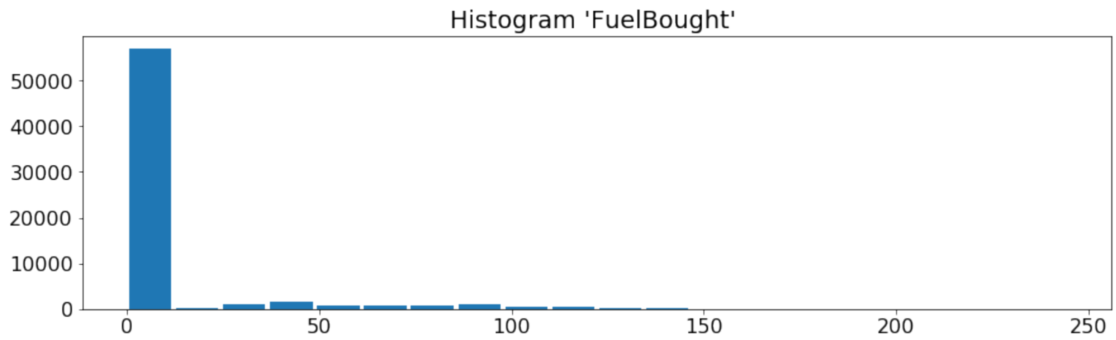
In [18]:
```
df3.describe().table()
```

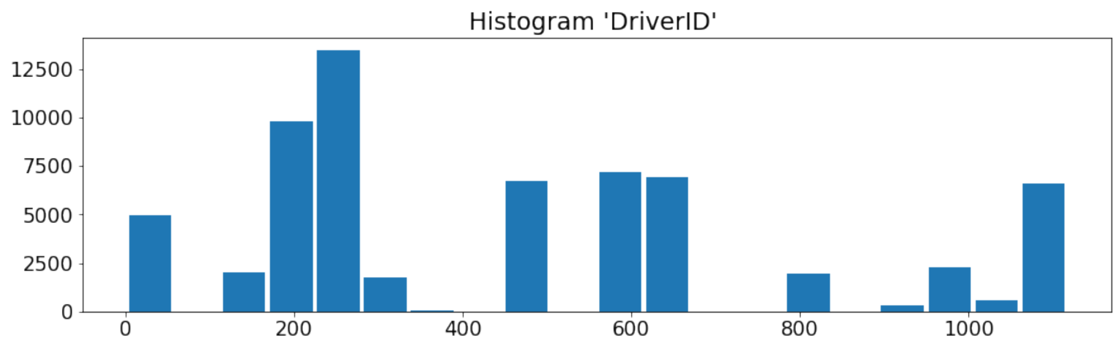
```
<IPython.core.display.HTML object>
```

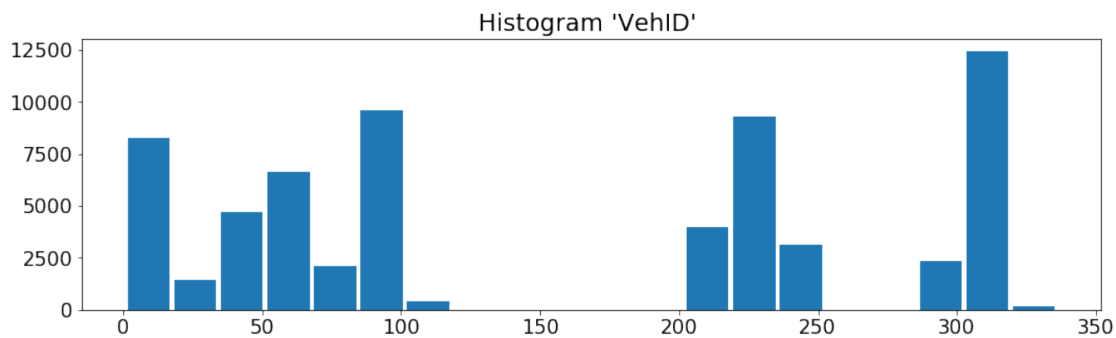
2.2.2 Histogram of columns

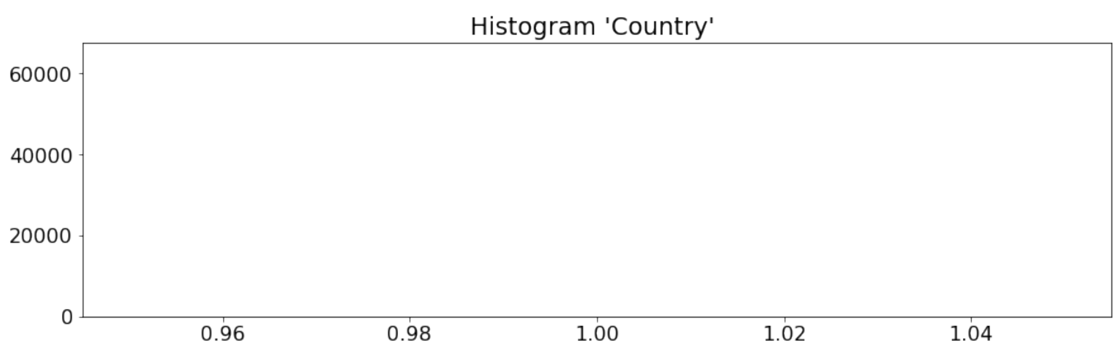
In [19]:
```
df1.plot.hist("*", 20)
```



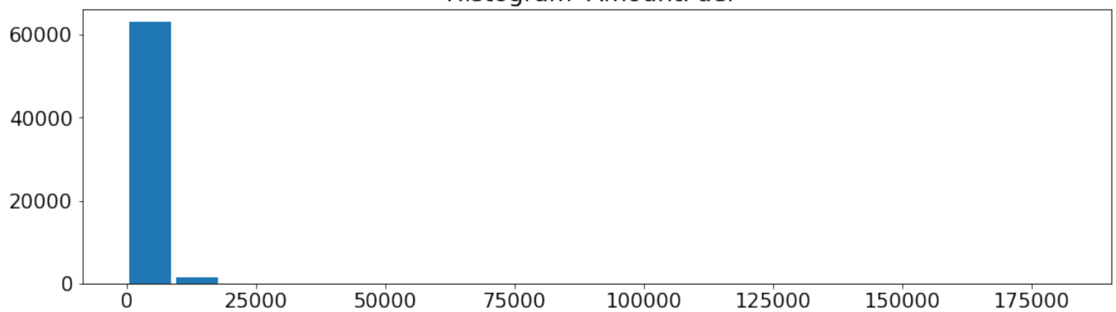

Histogram 'FuelBought'
50000
40000
30000
20000
10000
0
0
50
100
150
200
250


Histogram 'DriverID'
12500
10000
7500
5000
2500
0
0
200
400
600
800
1000

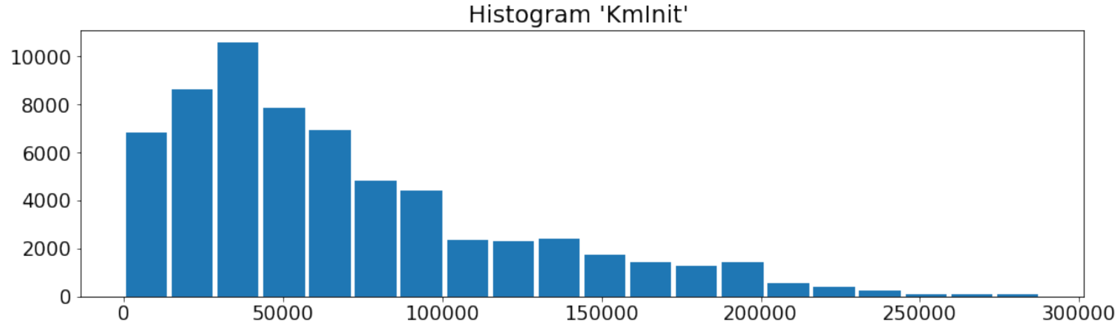

Histogram 'VehID'
12500
10000
7500
5000
2500
0
0
50
100
150
200
250
300
350


Histogram 'Country'
60000
40000
20000
0
0.96
0.98
1.00
1.02
1.04

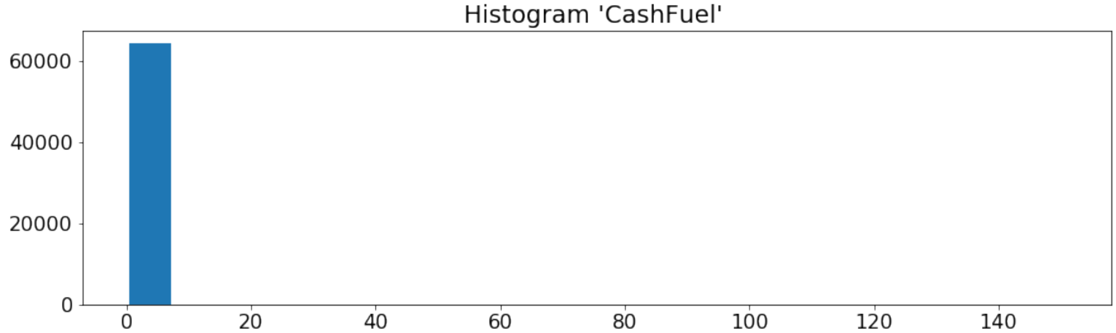
Histogram 'AmountFuel'

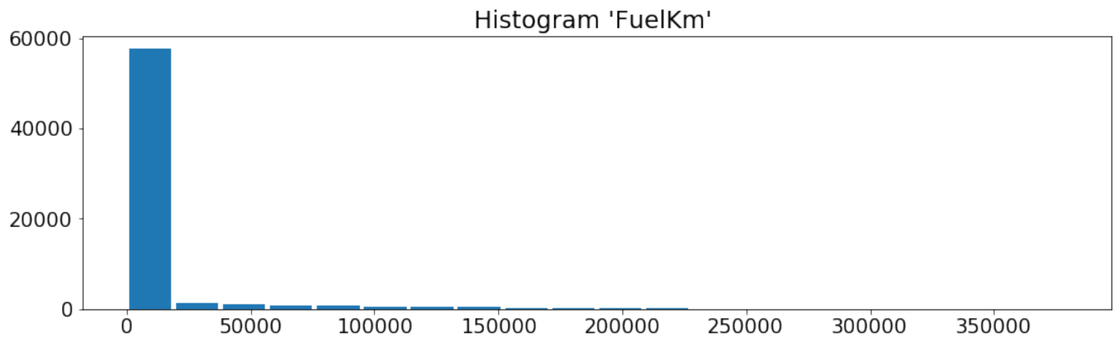


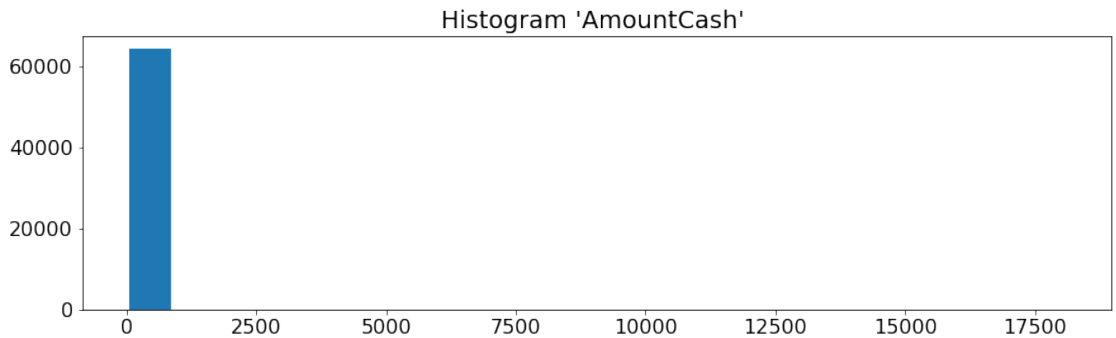
Histogram 'KmInit'



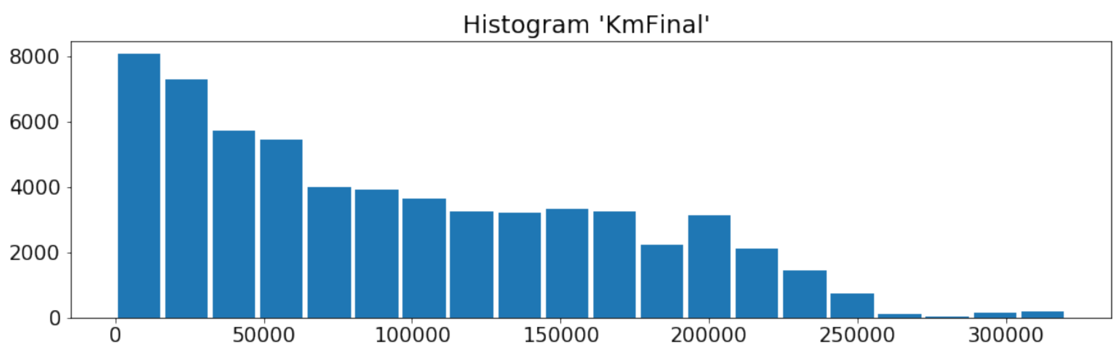
Histogram 'CashFuel'

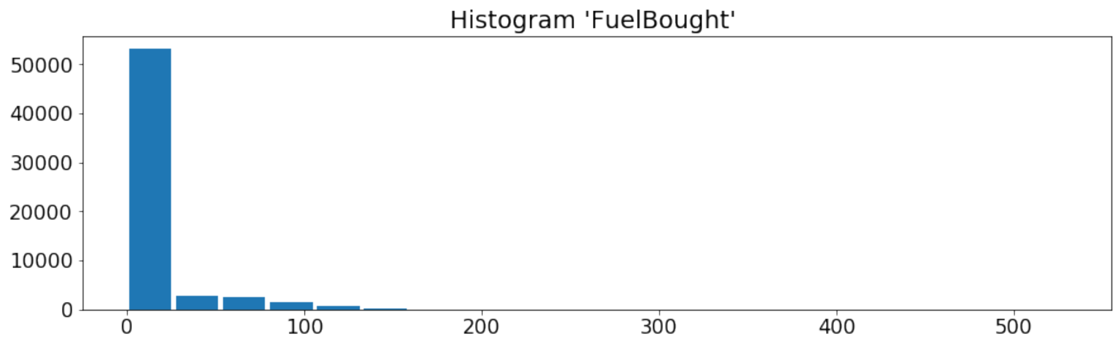


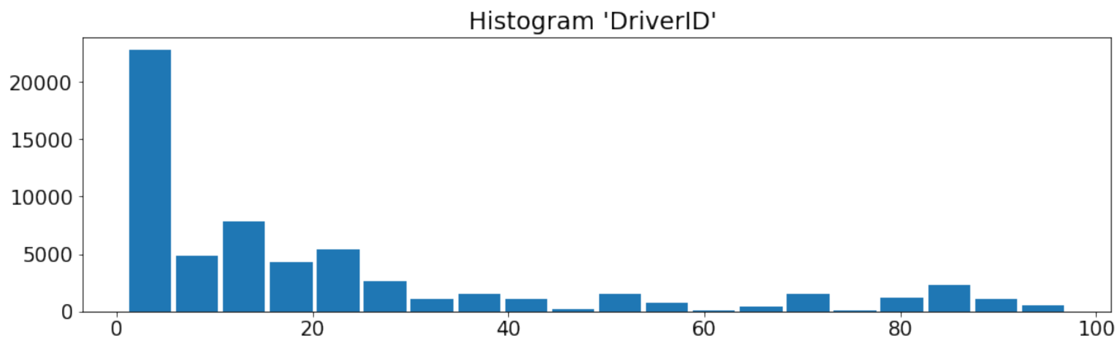


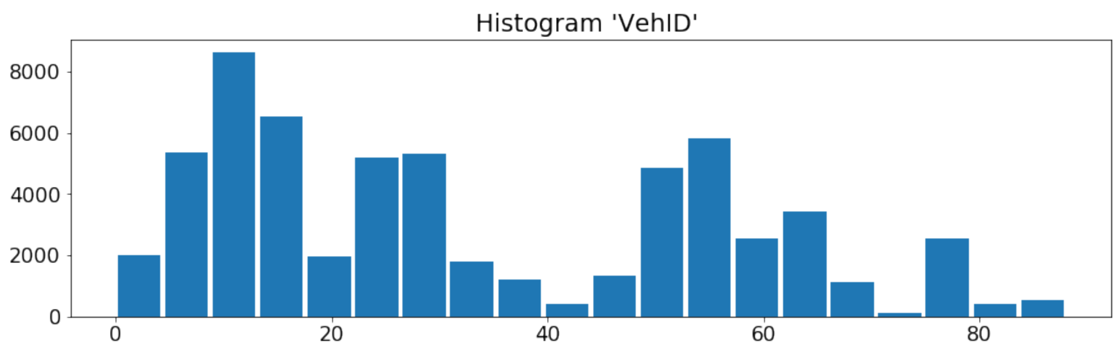


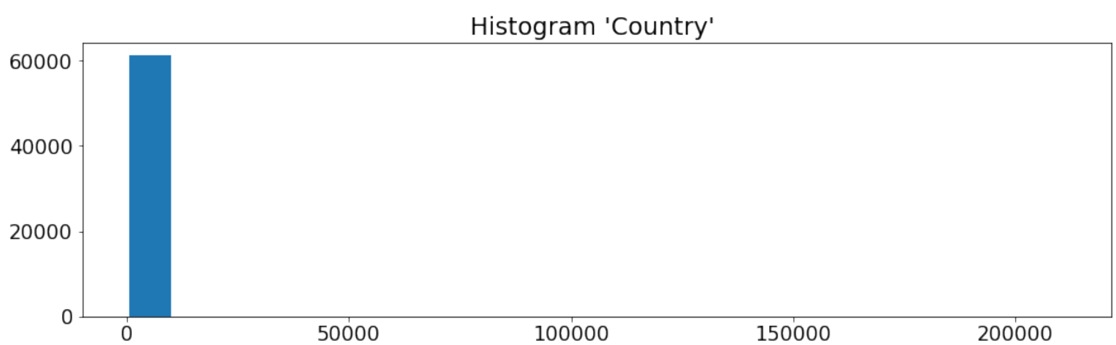

```
In [20]: df2.plot.hist("*", 20)
```

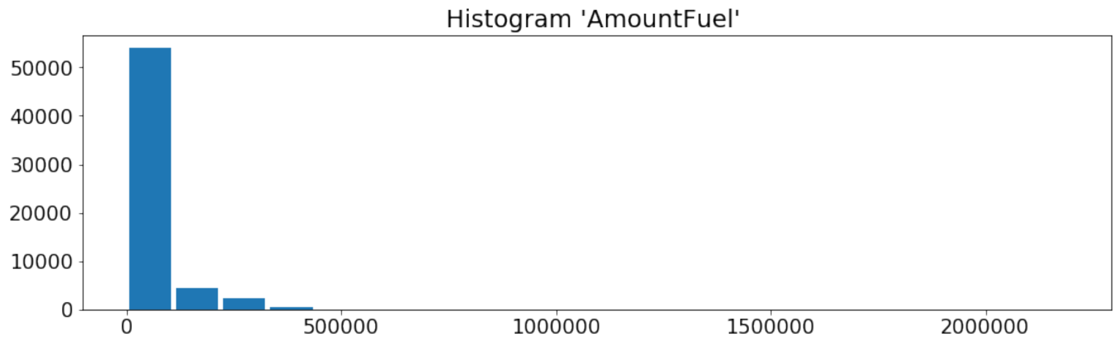


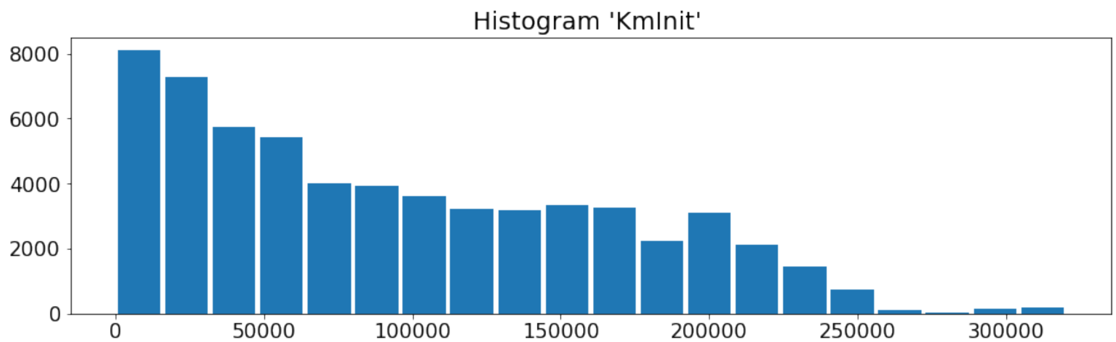

Histogram 'FuelBought'
0
10000
20000
30000
40000
50000
0
100
200
300
400
500

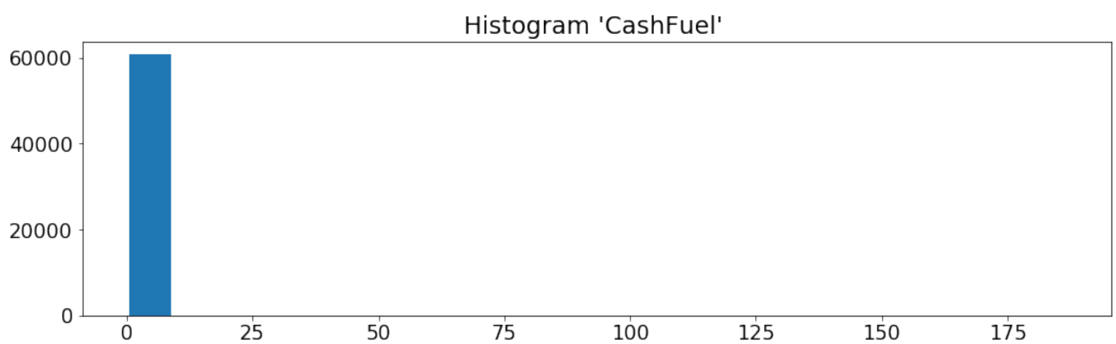

Histogram 'DriverID'
0
5000
10000
15000
20000
0
20
40
60
80
100

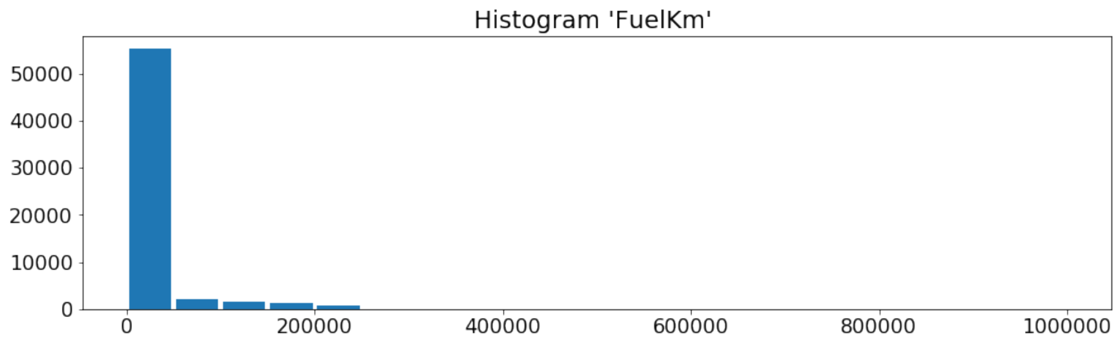

Histogram 'VehID'
0
2000
4000
6000
8000
0
20
40
60
80

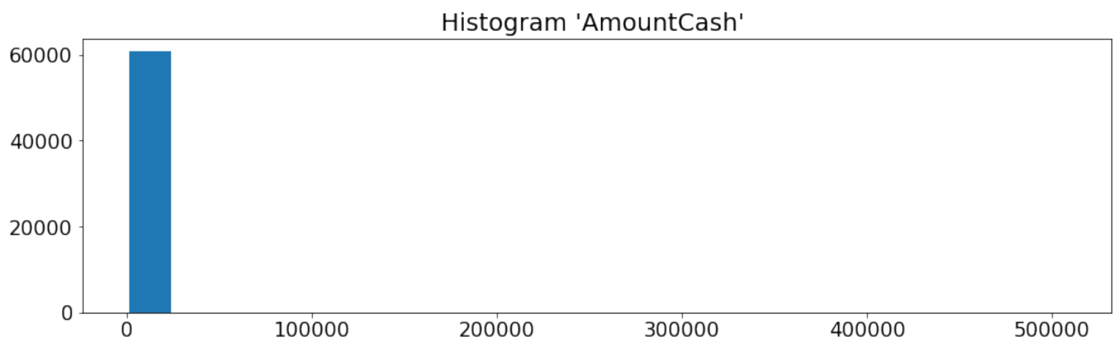

Histogram 'Country'
0
20000
40000
60000
0
50000
100000
150000
200000


Histogram 'AmountFuel'
50000
40000
30000
20000
10000
0
0
500000
1000000
1500000
2000000

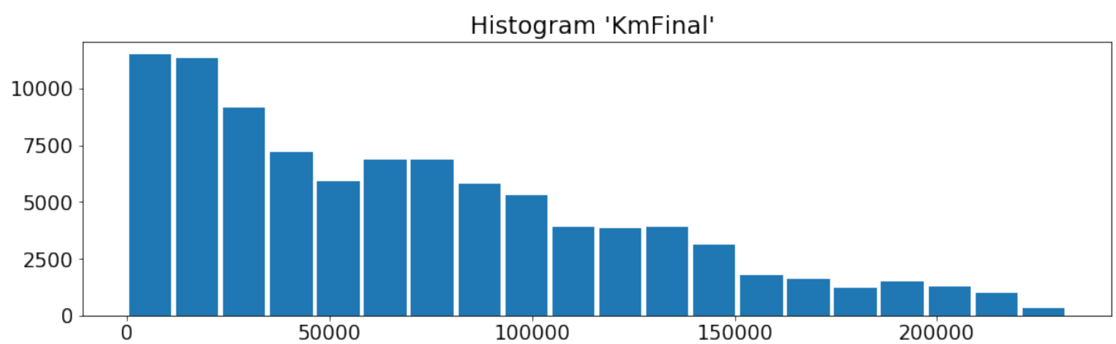

Histogram 'KmInit'
8000
6000
4000
2000
0
0
50000
100000
150000
200000
250000
300000

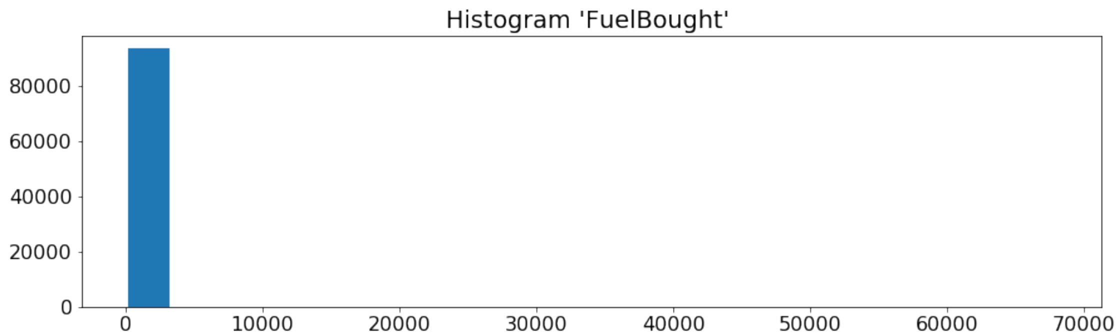

Histogram 'CashFuel'
60000
40000
20000
0
0
25
50
75
100
125
150
175

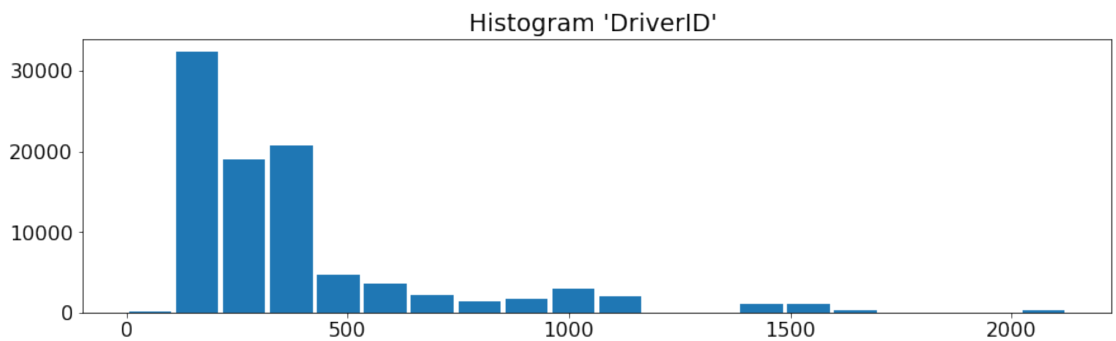


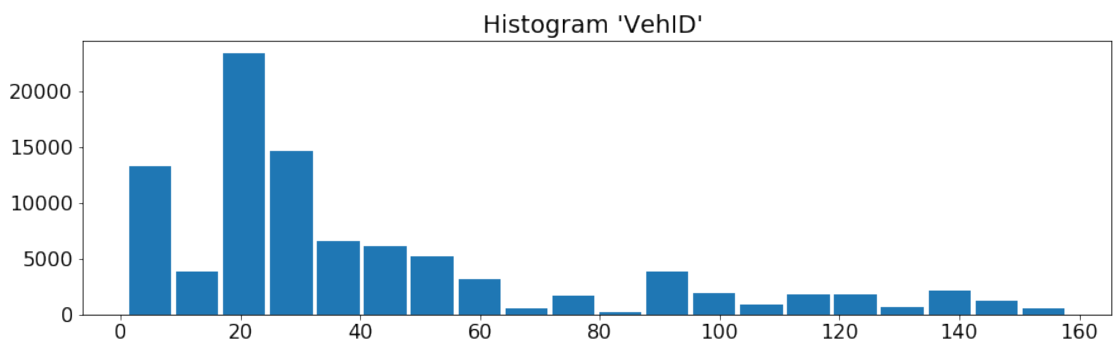


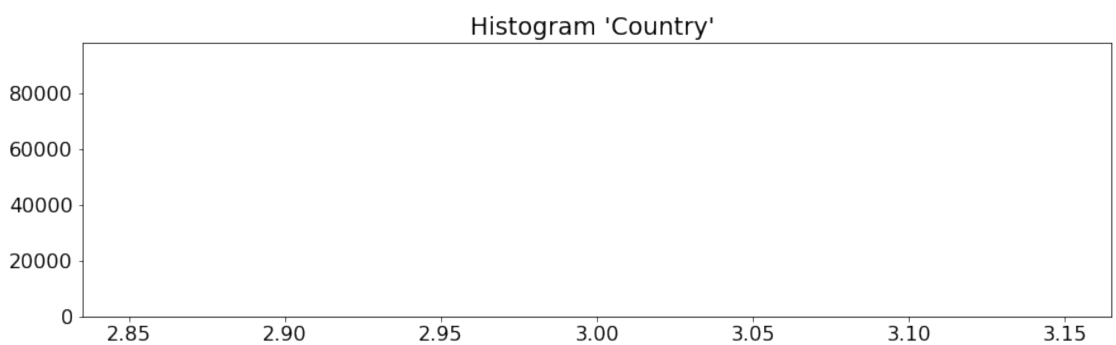

```
In [21]: df3.plot.hist("*", 20)
```

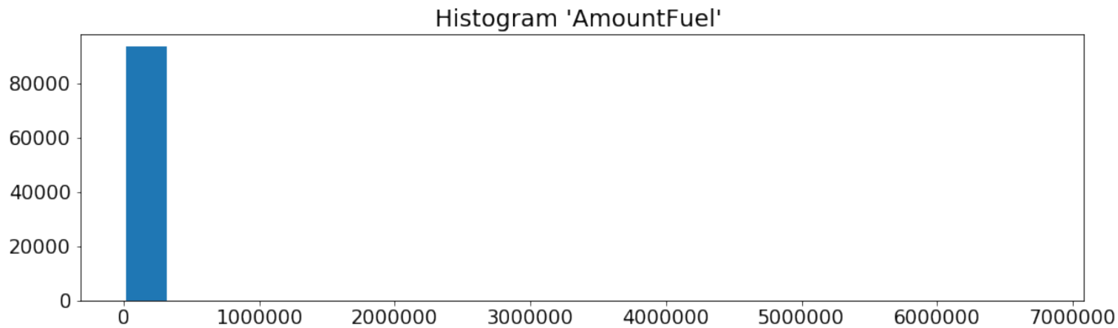


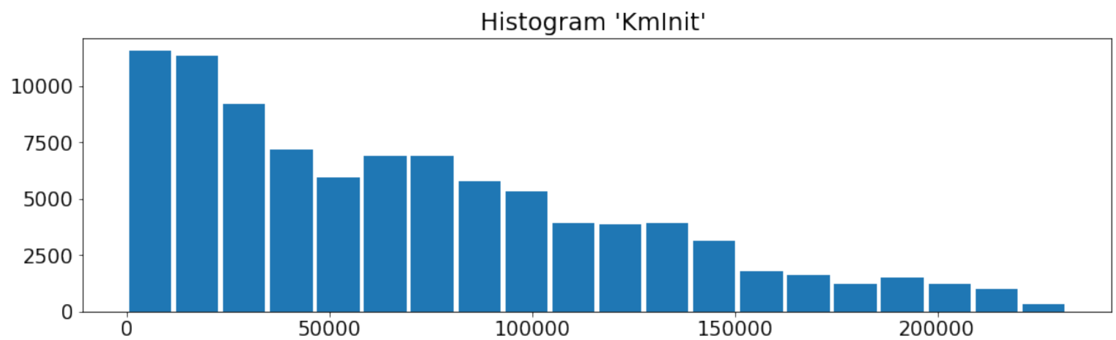

Histogram 'FuelBought'
80000
60000
40000
20000
0
0
10000
20000
30000
40000
50000
60000
70000

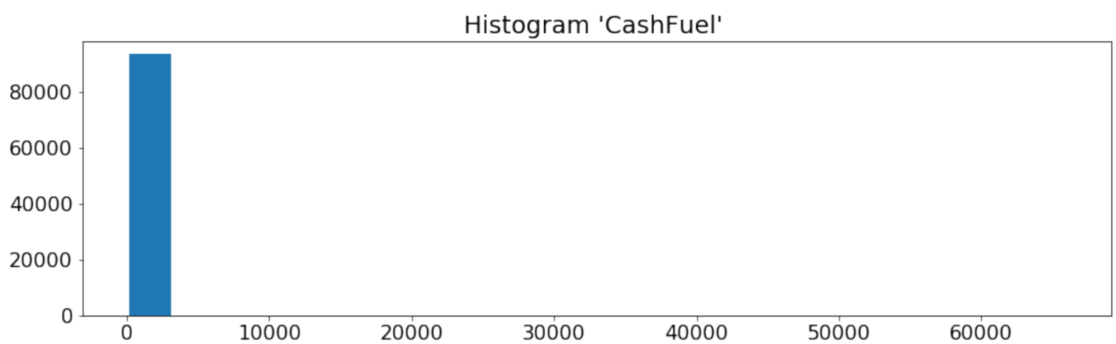

Histogram 'DriverID'
30000
20000
10000
0
0
500
1000
1500
2000

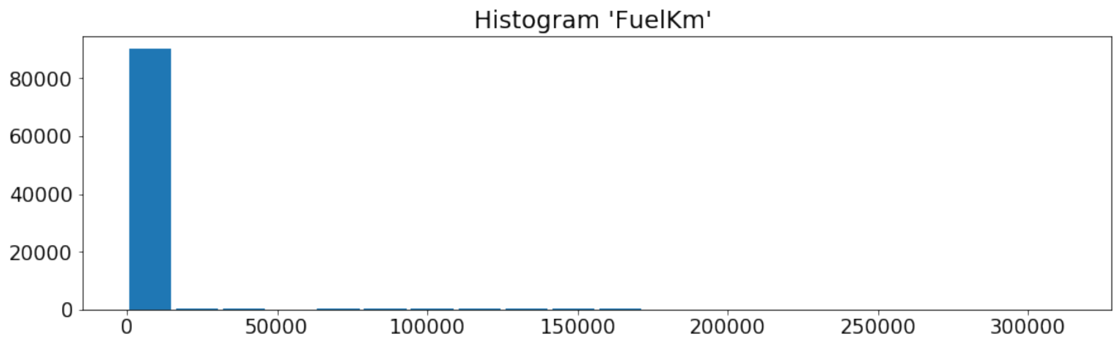

Histogram 'VehID'
20000
15000
10000
5000
0
0
20
40
60
80
100
120
140
160

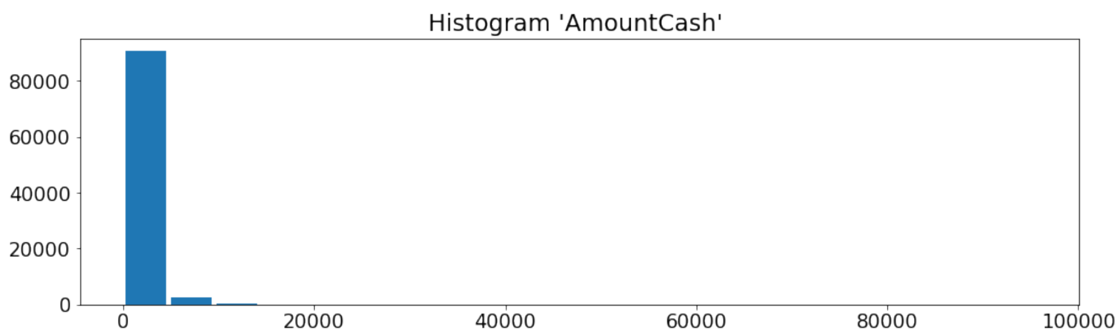

Histogram 'Country'
80000
60000
40000
20000
0
2.85
2.90
2.95
3.00
3.05
3.10
3.15

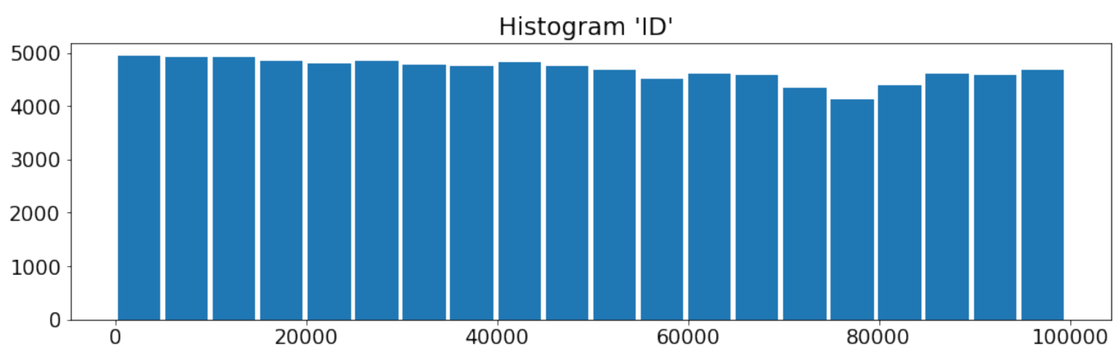

Histogram 'AmountFuel'
80000
60000
40000
20000
0
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000


Histogram 'KmInit'
10000
7500
5000
2500
0
0
50000
100000
150000
200000


Histogram 'CashFuel'
80000
60000
40000
20000
0
0
10000
20000
30000
40000
50000
60000







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 [22]: # Print the value counts for specific columns
         print('\nColumn Name:', df1.select("Destination").dtypes)
         df1.groupBy('Destination').count().rows.sort("count", "desc").table()
         # or I can use following command to do the same thing
         # df1.groupBy('Destination').count().orderBy("count", ascending=False).table())
```

```
Column Name: [('Destination', 'string')]
```

```
<IPython.core.display.HTML object>
```

```
In [23]: # Print the value counts for specific columns
         print('\nColumn Name:', df2.select("Destination").dtypes)
         df2.groupBy('Destination').count().rows.sort("count", "desc").table()
```

```
Column Name: [('Destination', 'string')]
```

```
<IPython.core.display.HTML object>
```

```
In [24]: # Print the value counts for specific columns
         print('\nColumn Name:', df3.select("Destination").dtypes)
         df3.groupBy('Destination').count().rows.sort("count", "desc").table()
```

```
Column Name: [('Destination', 'string')]
```

```
<IPython.core.display.HTML object>
```

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

If Spark is running on Hadoop cluster with multiple server nodes, the sample data set can combine the whole data sets from three files. For this project, the Spark is running on single node cluster. So, I will take 10,000 records from data sets so that it can take acceptable time to produce output.

```
In [25]: df1_sample = df1.sample_n(3000)
         df2_sample = df2.sample_n(3000)
         df3_sample = df3.sample_n(4000)
```

3.1.2 Combine three data sets

```
In [26]: df4 = df1_sample.unionAll(df2_sample).unionAll(df3_sample)
         print("Combined data set - df4: (", df4.count(), ",", df4.cols.count(), ")")
         df4.summary().table()
```

```
Combined data set - df4: ( 10262 , 13 )


<IPython.core.display.HTML object>
```

3.2 Create Columns based on Existing Ones

3.2.1 Calculate distance for every driving history record

```
In [27]: df4 = df4.withColumn("Distance", (df4.KmFinal - df4.KmInit))
```

3.2.2 Determine the directions of driving vehicle

```
In [28]: 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

    udf_usage_type = udf(usage_type, IntegerType())
    df4 = df4.withColumn("Directions", udf_usage_type(df4['Destination']))
```

3.2.3 Calculate how many stops during one driving history

```
In [29]: 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

         udf_get_stops = udf(get_stops, IntegerType())
         df4 = df4.withColumn("Stops", udf_get_stops(df4.Destination))
```

3.2.4 Calculate how fuel was added and purchased

```
In [30]: udf_when_fuel_added = udf(lambda x, y: x - y if (x >= y) else x, IntegerType())
         df4 = df4.withColumn("When_Fuel_Added", udf_when_fuel_added(df4.FuelKm, df4.KmInit))

         udf_is_fuel_added = udf(lambda x, y: 1 if (x > 0) or (y > 0) else 0, IntegerType())
         df4 = df4.withColumn("is_Fuel_Added", udf_is_fuel_added(df4.AmountFuel, df4.AmountCas

         udf_is_cash_paid = udf(lambda x: 1 if (x > 0) else 0, IntegerType())
         df4 = df4.withColumn("is_Cash_Paid", udf_is_cash_paid(df4.AmountCash))
```

3.3 Extract Data Set for Machine Learning Use

3.3.1 Query data set

```
In [31]: df = df4.where(df4.Distance >0).select("Country", "Directions", "Distance", "Stops", "
         # convert spark dataframe into pandas dataframe
         df_pd = df.toPandas()
         df.table(5)
         df_pd.head(5)
```

```
<IPython.core.display.HTML object>
```

```
Out[31]:    Country  Directions  Distance  Stops  is_Fuel_Added  is_Cash_Paid  \
         0        1           2        27      3              0             0
         1        1           1        35      2              0             0
         2        1           2         8      3              0             0
         3        1           2        14      3              0             0
         4        1           2        19      3              0             0

            When_Fuel_Added
         0                0
         1                0
         2                0
         3                0
         4                0
```

3.3.2 Data Set Profiling

```
In [86]: op.profiler.run(df, "*",infer=False)
```

```
<IPython.core.display.HTML object>


ERROR:pika.adapters.base_connection:Connection to 127.0.0.1:5672 failed: [Errno 111] Connection
WARNING:pika.connection:Could not connect, 0 attempts left
ERROR:pika.adapters.blocking_connection:Connection open failed - 'Connection to 127.0.0.1:5672



        ---------------------------------------------------------------------------

        ConnectionClosed                          Traceback (most recent call last)

        <ipython-input-86-3c4588992e63> in <module>
    ----> 1 op.profiler.run(df, "*",infer=False)


        /usr/local/lib/python3.6/dist-packages/optimus/helpers/decorators.py in timed(*args, **
         25     def timed(*args, **kw):
```

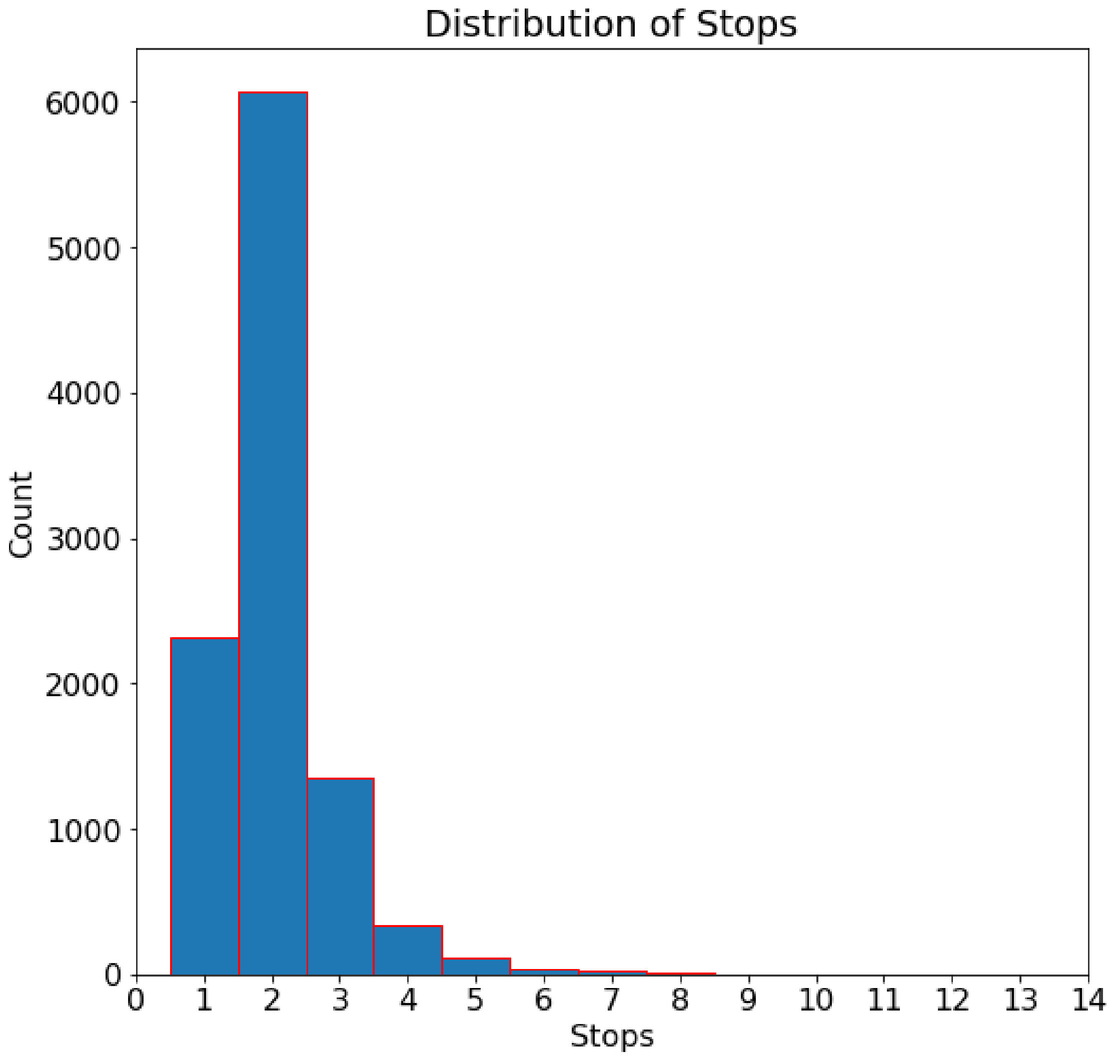
```
     26         start_time = timeit.default_timer()
---> 27         f = method(*args, **kw)
     28         _time = round(timeit.default_timer() - start_time, 2)
     29         logging.info("{name}() executed in {time} sec".format(name=method.__name__


/usr/local/lib/python3.6/dist-packages/optimus/profiler/profiler.py in run(self, df, c
    237
    238         if self.queue_url is not None:
--> 239             self.to_queue(output)
    240
    241         # Save to file


/usr/local/lib/python3.6/dist-packages/optimus/profiler/profiler.py in to_queue(self, 
    251         url = os.environ.get('CLOUDAMQP_URL', self.queue_url)
    252         params = pika.URLParameters(url)
--> 253         connection = pika.BlockingConnection(params)
    254         channel = connection.channel()  # start a channel
    255         channel.queue_declare(queue='optimus')  # Declare a queue


/usr/local/lib/python3.6/dist-packages/pika/adapters/blocking_connection.py in __init_
    375         self._impl.ioloop.activate_poller()
    376
--> 377         self._process_io_for_connection_setup()
    378
    379     def __repr__(self):


/usr/local/lib/python3.6/dist-packages/pika/adapters/blocking_connection.py in _process
    415         if not self._open_error_result.ready:
    416             self._flush_output(self._opened_result.is_ready,
--> 417                                self._open_error_result.is_ready)
    418
    419         if self._open_error_result.ready:


/usr/local/lib/python3.6/dist-packages/pika/adapters/blocking_connection.py in _flush_
    469                             raise maybe_exception
    470                         else:
--> 471                             raise exceptions.ConnectionClosed(maybe_exception)
    472                 else:
    473                         result = self._closed_result.value


ConnectionClosed: Connection to 127.0.0.1:5672 failed: [Errno 111] Connection refused
```

3.4 Data Exploration for Trending

3.4.1 Column distributions

In [34]:
```
# Bar plot of Stops
plt.bar(df_pd['Stops'].value_counts().index, df_pd['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)))
```

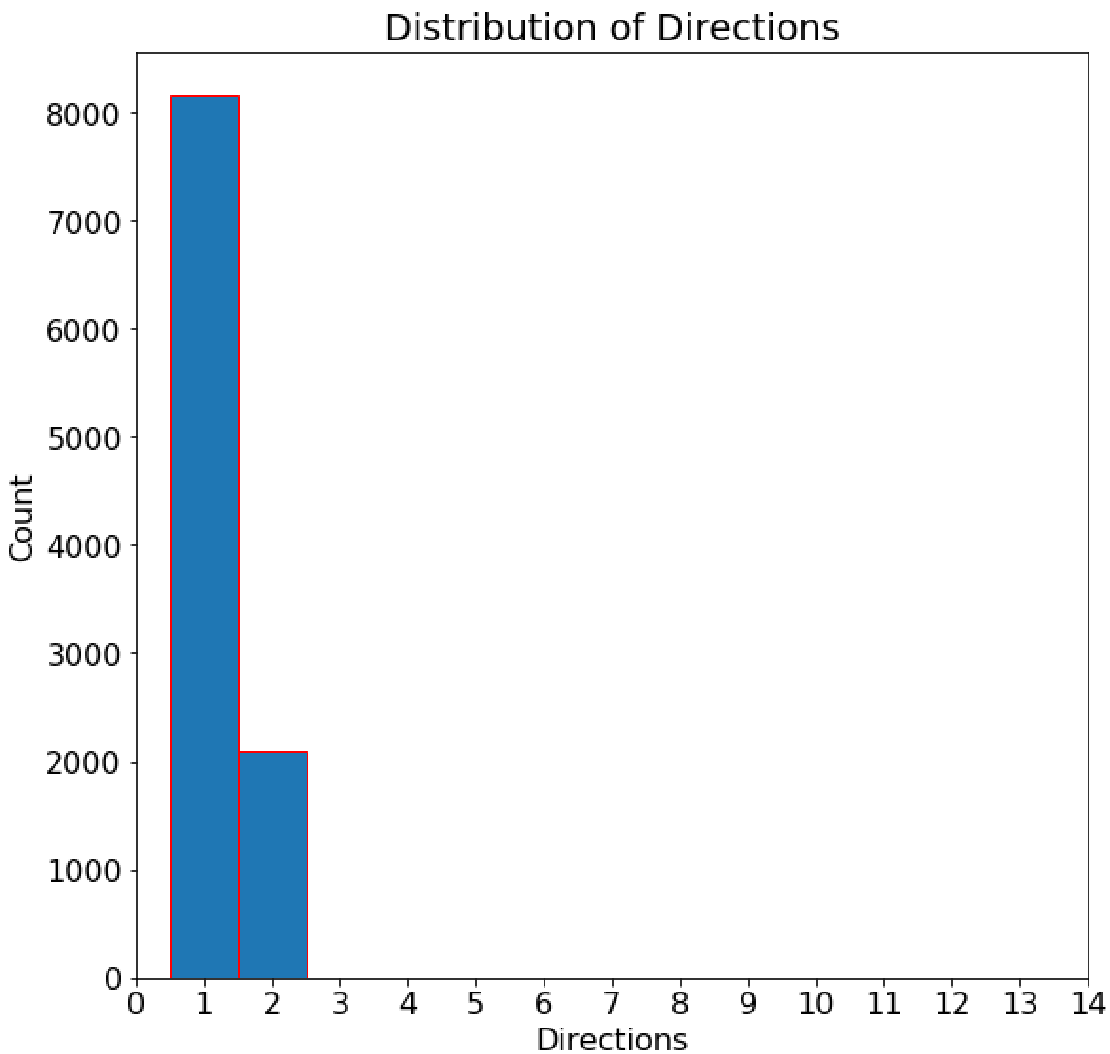
Out[34]:
```
([<matplotlib.axis.XTick at 0x7fa6744eac18>,
  <matplotlib.axis.XTick at 0x7fa67d1b7eb8>,
  <matplotlib.axis.XTick at 0x7fa67d1b7be0>,
  <matplotlib.axis.XTick at 0x7fa67cf4c128>,
  <matplotlib.axis.XTick at 0x7fa67cf4cac8>,
  <matplotlib.axis.XTick at 0x7fa67d29ea58>,
  <matplotlib.axis.XTick at 0x7fa67cee4c18>,
  <matplotlib.axis.XTick at 0x7fa67cee4748>,
  <matplotlib.axis.XTick at 0x7fa67cee43c8>,
  <matplotlib.axis.XTick at 0x7fa67d11f518>,
  <matplotlib.axis.XTick at 0x7fa67d11fa20>,
  <matplotlib.axis.XTick at 0x7fa67d11f4e0>,
  <matplotlib.axis.XTick at 0x7fa67cf85b38>,
  <matplotlib.axis.XTick at 0x7fa67d11fb38>,
  <matplotlib.axis.XTick at 0x7fa67cee4550>],
 <a list of 15 Text xticklabel objects>)
```




```
In [35]: # Bar plot of Directions
         plt.bar(df_pd['Directions'].value_counts().index, df_pd['Directions'].value_counts().v
                 fill = 'blue', edgecolor = 'r', width = 1)
         plt.xlabel('Directions')
         plt.ylabel('Count')
         plt.title('Distribution of Directions')
         plt.xticks(list(range(0, 15)))
```

```
Out[35]: ([<matplotlib.axis.XTick at 0x7fa67d0c28d0>,
           <matplotlib.axis.XTick at 0x7fa67cf25748>,
           <matplotlib.axis.XTick at 0x7fa67cf25390>,
           <matplotlib.axis.XTick at 0x7fa67411fcf8>,
           <matplotlib.axis.XTick at 0x7fa67ce04320>,
           <matplotlib.axis.XTick at 0x7fa67ce047f0>,
           <matplotlib.axis.XTick at 0x7fa67ce04cf8>,
```

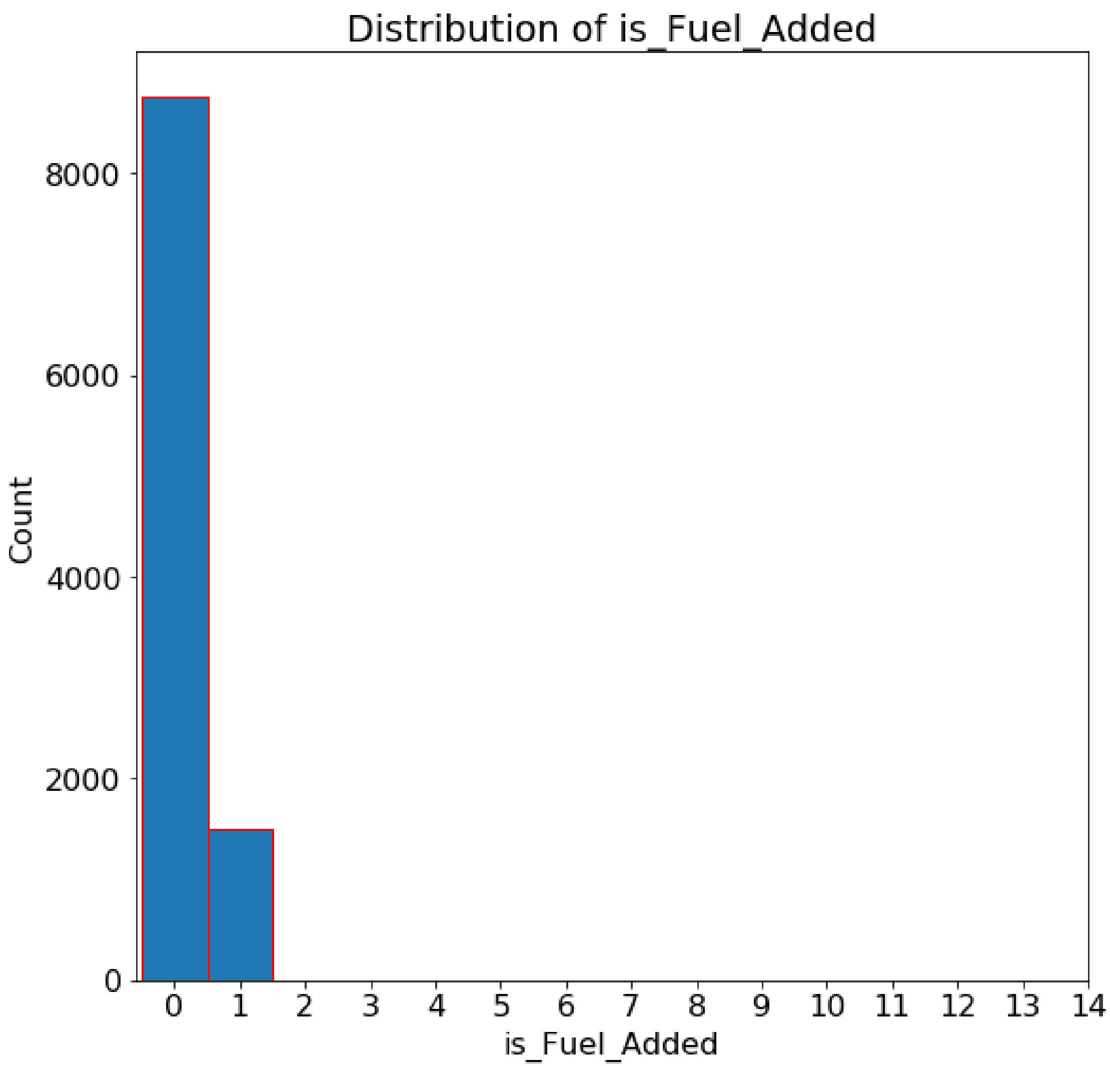
```
 <matplotlib.axis.XTick at 0x7fa67cdf9240>,
 <matplotlib.axis.XTick at 0x7fa67cdf9748>,
 <matplotlib.axis.XTick at 0x7fa67cdf9c50>,
 <matplotlib.axis.XTick at 0x7fa67c0e6400>,
 <matplotlib.axis.XTick at 0x7fa67cdf9ac8>,
 <matplotlib.axis.XTick at 0x7fa67ce04400>,
 <matplotlib.axis.XTick at 0x7fa67c0e63c8>,
 <matplotlib.axis.XTick at 0x7fa67c0e6ef0>],
<a list of 15 Text xticklabel objects>)
```



In [36]:
```
# Bar plot of is_Fuel_Added
plt.bar(df_pd['is_Fuel_Added'].value_counts().index, df_pd['is_Fuel_Added'].value_cou
        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)))
```

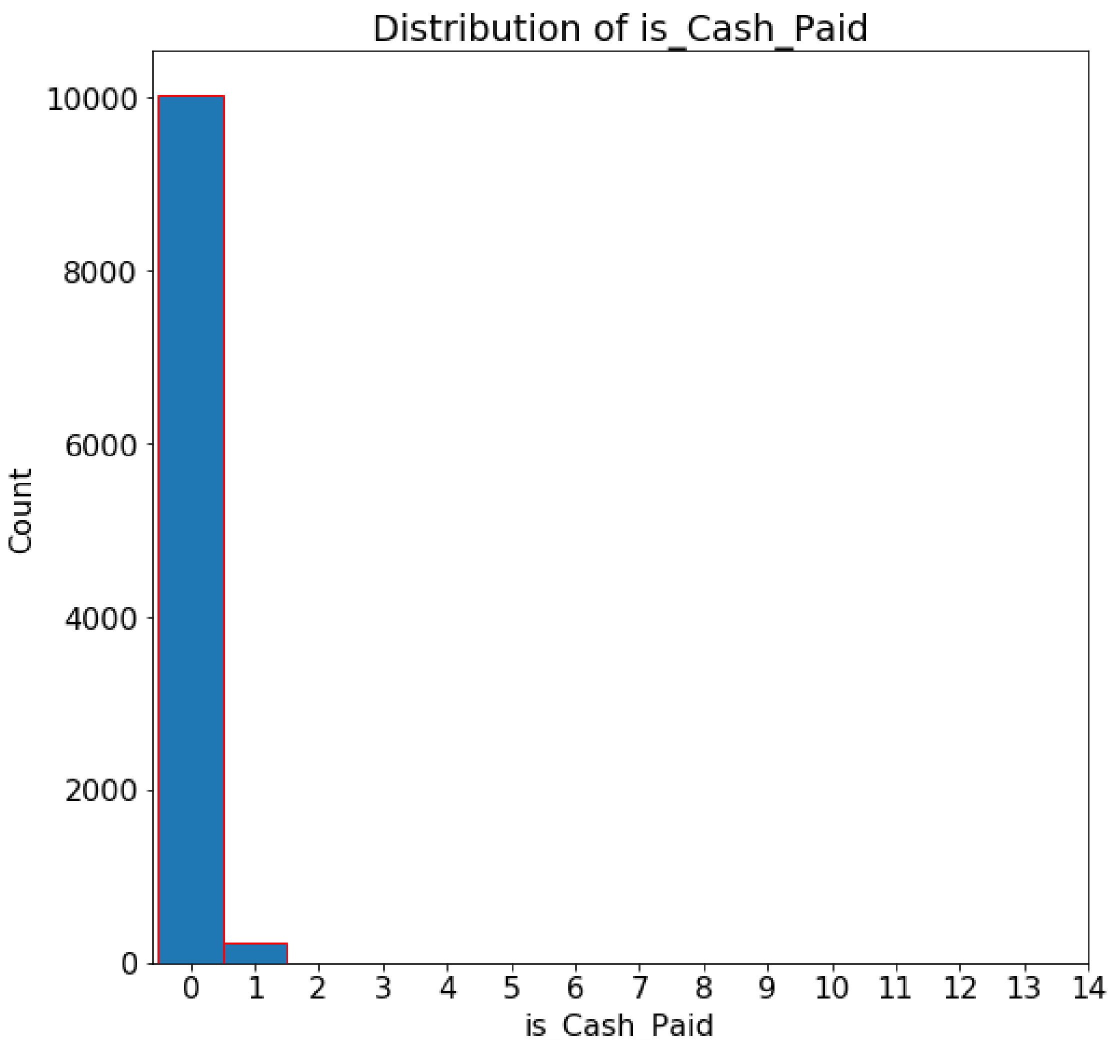
Out[36]:
```
([<matplotlib.axis.XTick at 0x7fa67d086748>,
  <matplotlib.axis.XTick at 0x7fa67d0860f0>,
  <matplotlib.axis.XTick at 0x7fa67d07cdd8>,
  <matplotlib.axis.XTick at 0x7fa67d0abb70>,
  <matplotlib.axis.XTick at 0x7fa67cf38208>,
  <matplotlib.axis.XTick at 0x7fa67cf386d8>,
  <matplotlib.axis.XTick at 0x7fa67cf38be0>,
  <matplotlib.axis.XTick at 0x7fa67cf60160>,
  <matplotlib.axis.XTick at 0x7fa67cf60630>,
  <matplotlib.axis.XTick at 0x7fa67cf60b38>,
  <matplotlib.axis.XTick at 0x7fa67cf590f0>,
  <matplotlib.axis.XTick at 0x7fa67cf60940>,
  <matplotlib.axis.XTick at 0x7fa67cf382b0>,
  <matplotlib.axis.XTick at 0x7fa67cf59518>,
  <matplotlib.axis.XTick at 0x7fa67cf59a20>],
 <a list of 15 Text xticklabel objects>)
```



In [37]:
```
# Bar plot of is_Cash_Paid
plt.bar(df_pd['is_Cash_Paid'].value_counts().index, df_pd['is_Cash_Paid'].value_counts
        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[37]:
```
([<matplotlib.axis.XTick at 0x7fa67c4b7240>,
  <matplotlib.axis.XTick at 0x7fa67c4ddb70>,
  <matplotlib.axis.XTick at 0x7fa67c4dd898>,
  <matplotlib.axis.XTick at 0x7fa67d1616a0>,
  <matplotlib.axis.XTick at 0x7fa67d161cf8>,
  <matplotlib.axis.XTick at 0x7fa67d162208>,
  <matplotlib.axis.XTick at 0x7fa67d1626d8>,
```

```
 <matplotlib.axis.XTick at 0x7fa67d162ba8>,
 <matplotlib.axis.XTick at 0x7fa67d159128>,
 <matplotlib.axis.XTick at 0x7fa67d1595c0>,
 <matplotlib.axis.XTick at 0x7fa67d159ac8>,
 <matplotlib.axis.XTick at 0x7fa67d162828>,
 <matplotlib.axis.XTick at 0x7fa67d159cf8>,
 <matplotlib.axis.XTick at 0x7fa67d159f60>,
 <matplotlib.axis.XTick at 0x7fa67c5da4a8>],
<a list of 15 Text xticklabel objects>)
```

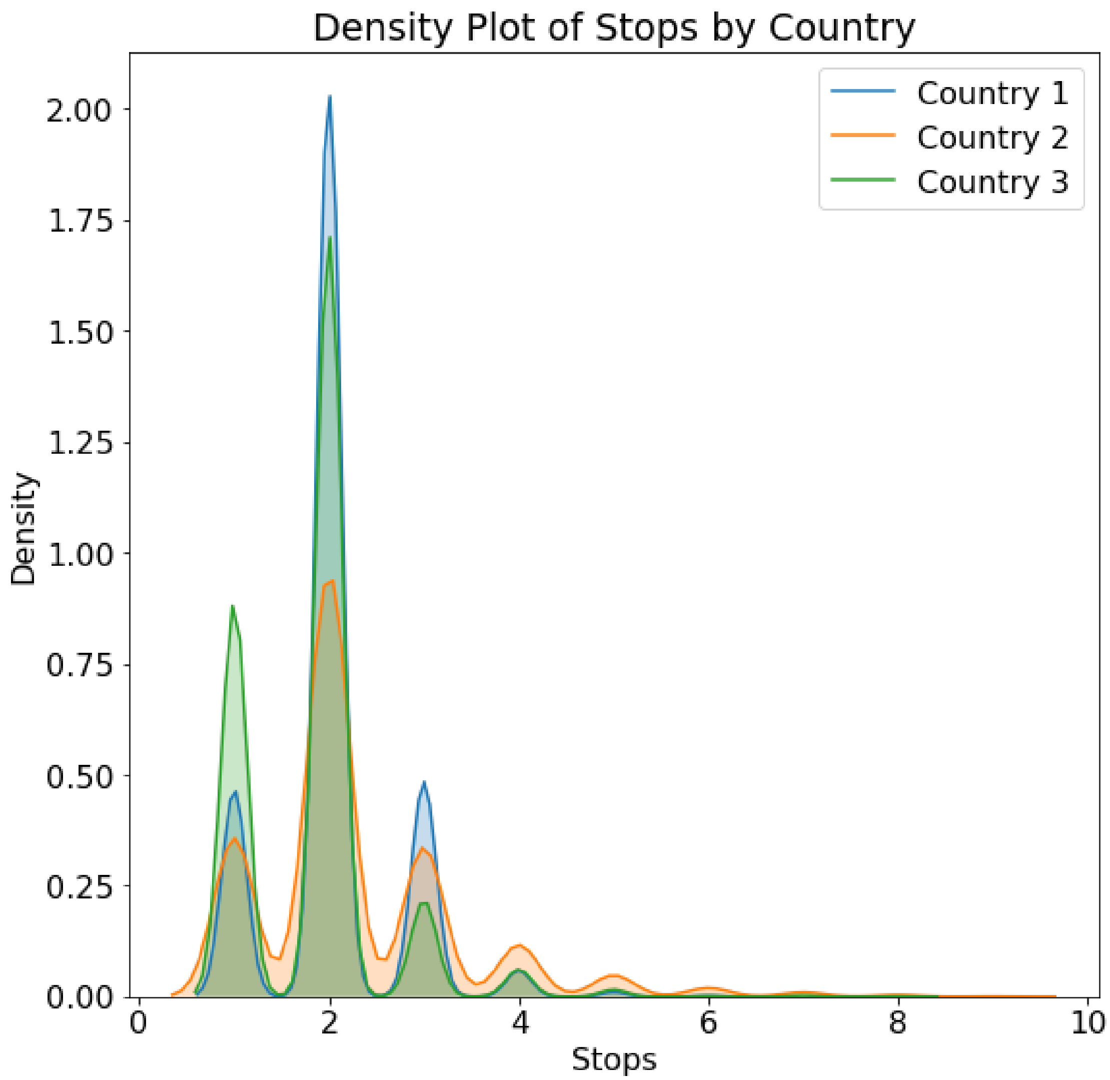


3.4.2 Column distributions by different countries

```
In [38]: # Stops distribution by Country
         sns.kdeplot(df_pd.loc[df_pd['Country'] == 1, 'Stops'], label = 'Country 1', shade = T
         sns.kdeplot(df_pd.loc[df_pd['Country'] == 2, 'Stops'], label = 'Country 2', shade = T
```

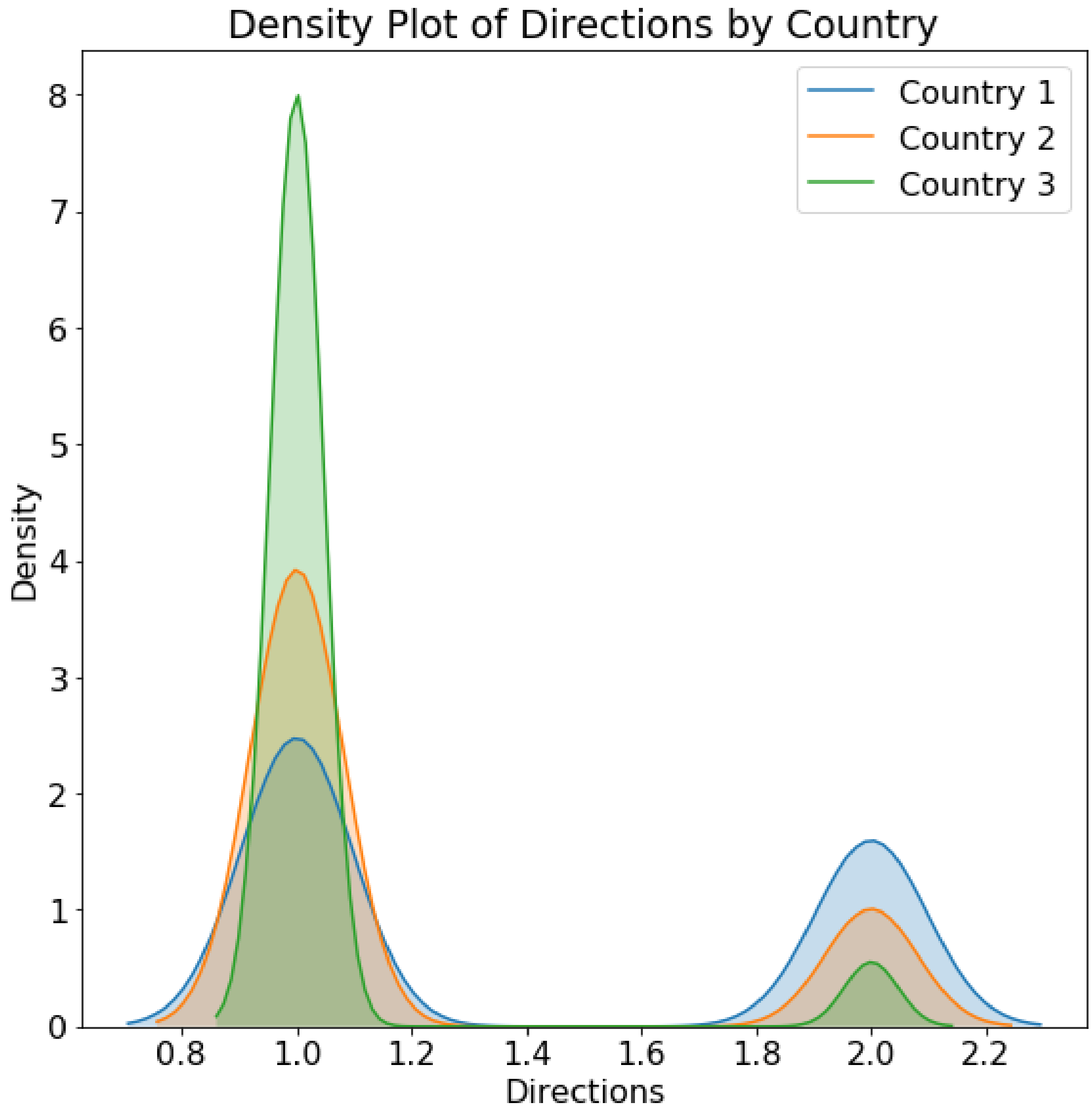
```
sns.kdeplot(df_pd.loc[df_pd['Country'] == 3, 'Stops'], label = 'Country 3', shade = Tr
plt.xlabel('Stops')
plt.ylabel('Density')
plt.title('Density Plot of Stops by Country')
```

Out[38]: Text(0.5, 1.0, 'Density Plot of Stops by Country')



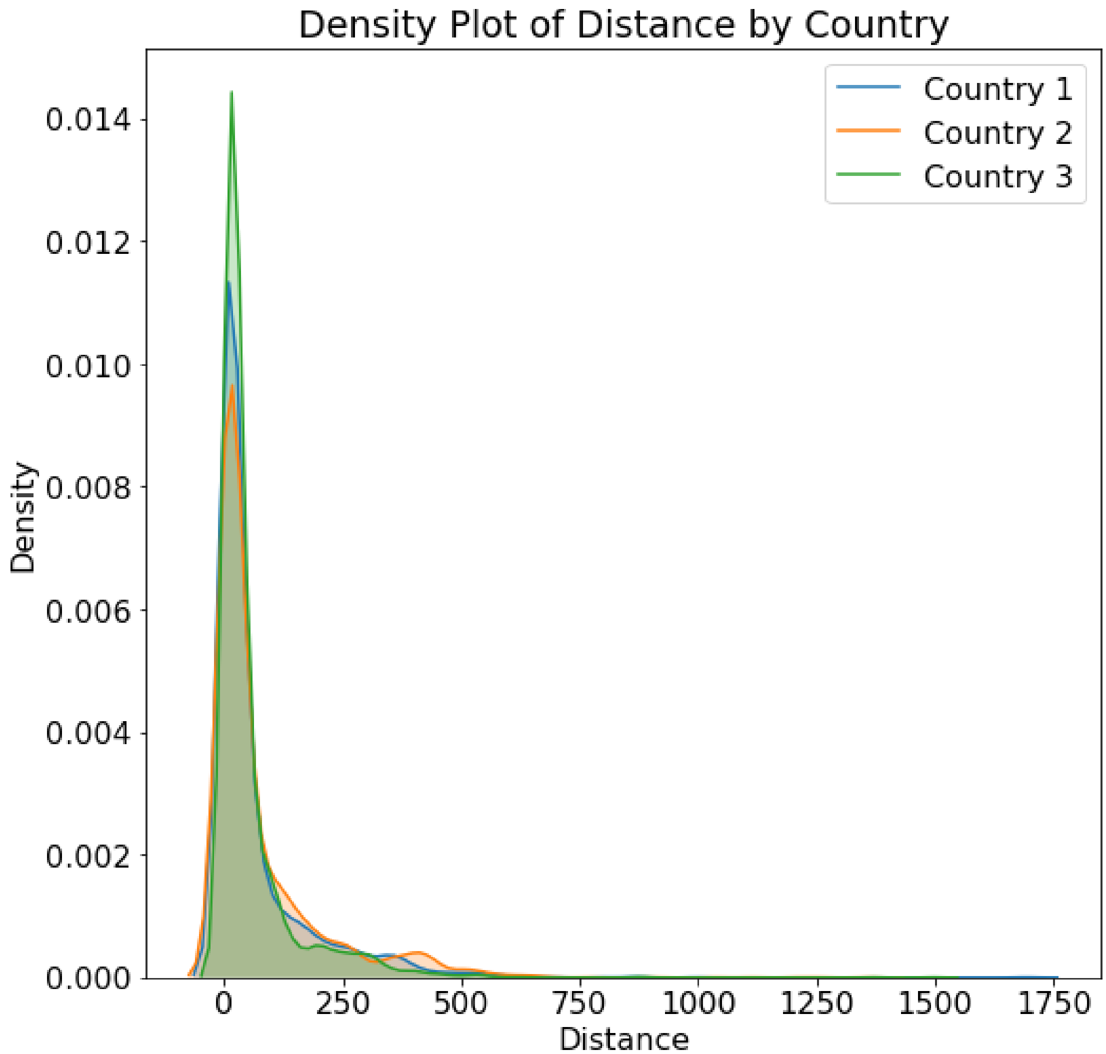
In [39]:
```
# Directions distribution by Country
sns.kdeplot(df_pd.loc[df_pd['Country'] == 1, 'Directions'], label = 'Country 1', shade
sns.kdeplot(df_pd.loc[df_pd['Country'] == 2, 'Directions'], label = 'Country 2', shade
sns.kdeplot(df_pd.loc[df_pd['Country'] == 3, 'Directions'], label = 'Country 3', shade
plt.xlabel('Directions')
plt.ylabel('Density')
plt.title('Density Plot of Directions by Country')
```

Out[39]: Text(0.5, 1.0, 'Density Plot of Directions by Country')

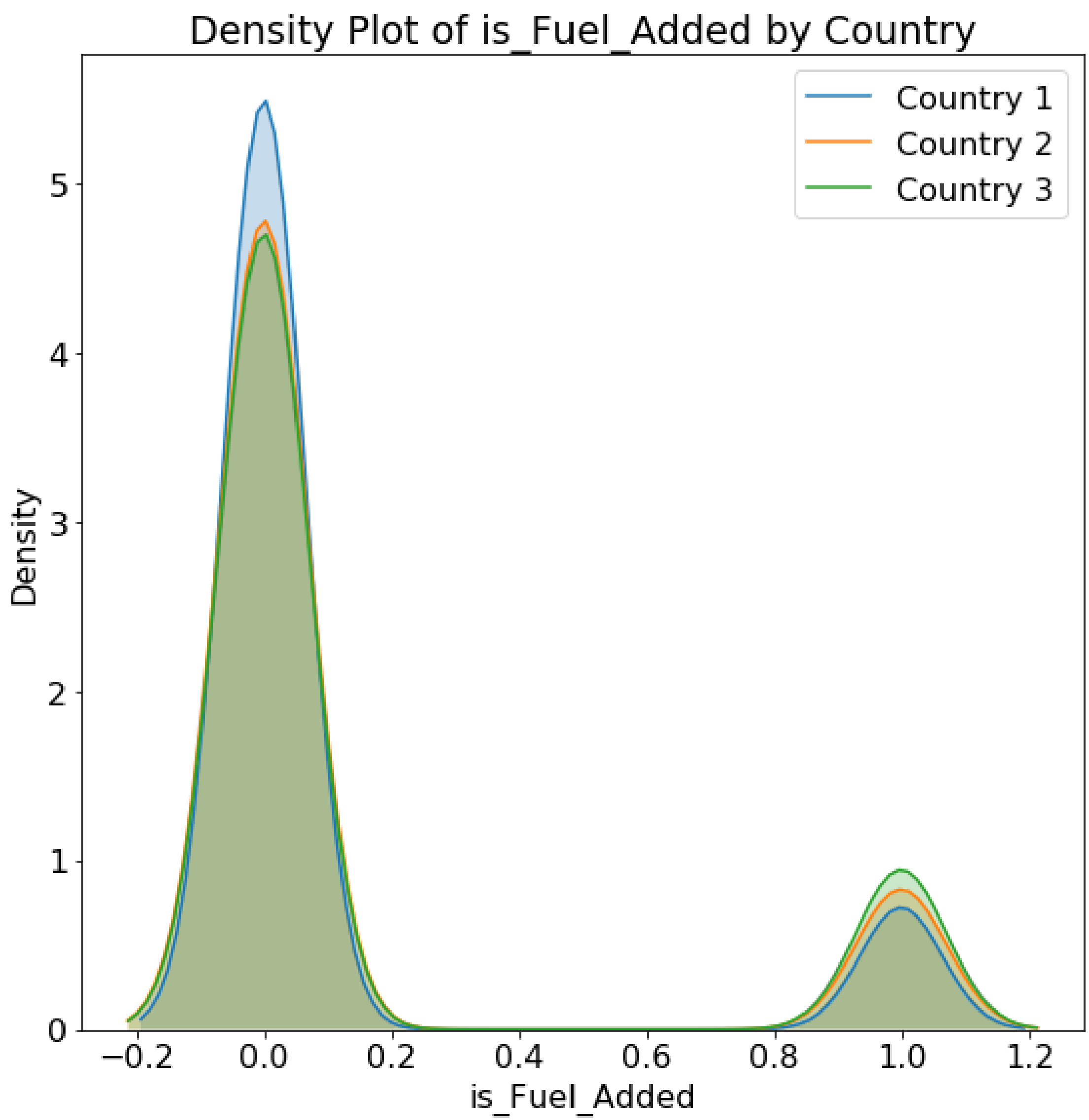



```
In [40]: # Distance distribution by Country
         sns.kdeplot(df_pd.loc[df_pd['Country'] == 1, 'Distance'], label = 'Country 1', shade =
         sns.kdeplot(df_pd.loc[df_pd['Country'] == 2, 'Distance'], label = 'Country 2', shade =
         sns.kdeplot(df_pd.loc[df_pd['Country'] == 3, 'Distance'], label = 'Country 3', shade =
         plt.xlabel('Distance')
         plt.ylabel('Density')
         plt.title('Density Plot of Distance by Country')
```

Out[40]: Text(0.5, 1.0, 'Density Plot of Distance by Country')

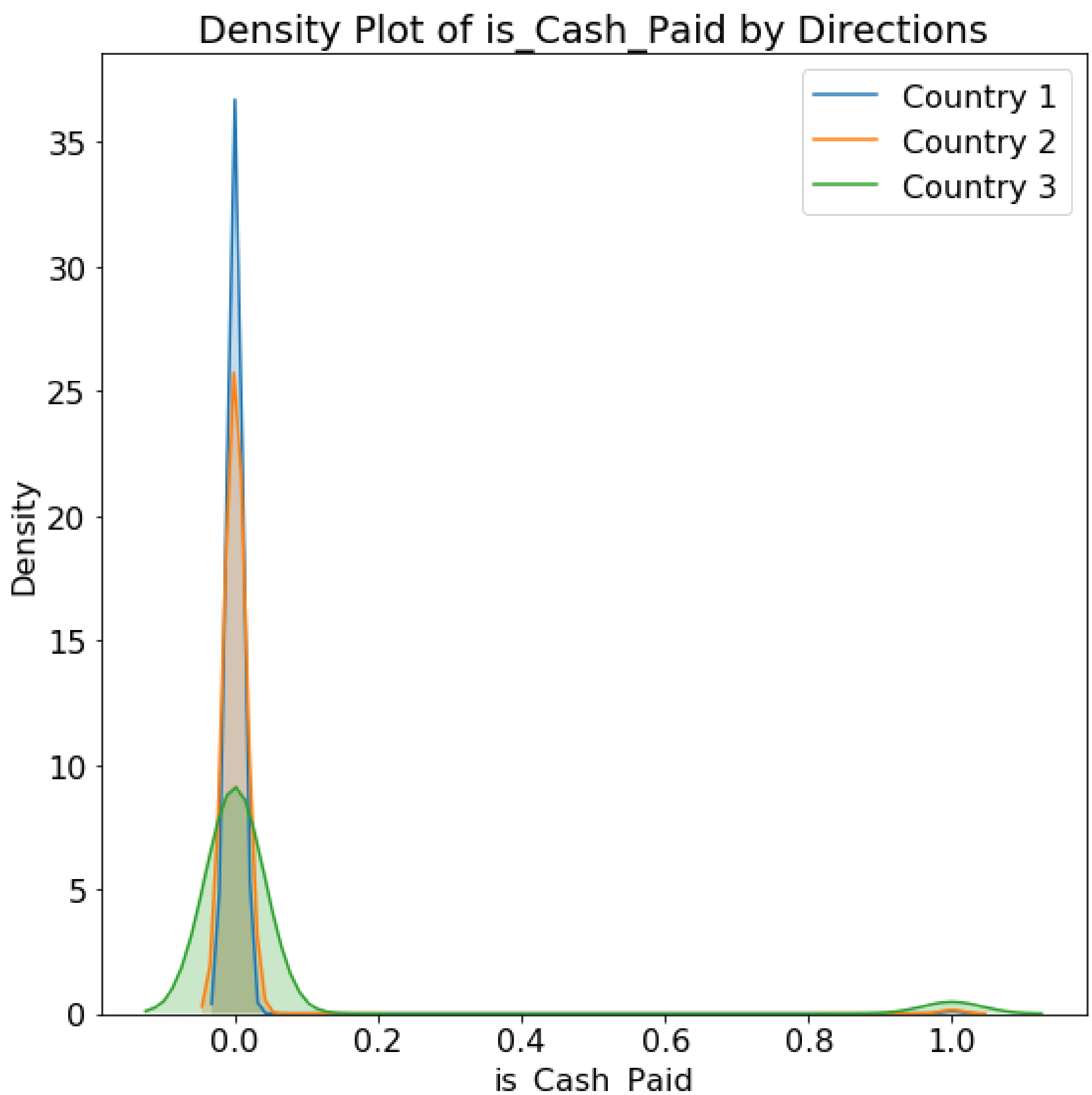



```
In [41]: # is_Fuel_Added distribution by Country
         sns.kdeplot(df_pd.loc[df_pd['Country'] == 1, 'is_Fuel_Added'], label = 'Country 1', sh
         sns.kdeplot(df_pd.loc[df_pd['Country'] == 2, 'is_Fuel_Added'], label = 'Country 2', sh
         sns.kdeplot(df_pd.loc[df_pd['Country'] == 3, 'is_Fuel_Added'], label = 'Country 3', sh
         plt.xlabel('is_Fuel_Added')
         plt.ylabel('Density')
         plt.title('Density Plot of is_Fuel_Added by Country')

Out[41]: Text(0.5, 1.0, 'Density Plot of is_Fuel_Added by Country')
```



In [42]:
```
# is_Cash_Paid distribution by Country
sns.kdeplot(df_pd.loc[df_pd['Country'] == 1, 'is_Cash_Paid'], label = 'Country 1', sha
sns.kdeplot(df_pd.loc[df_pd['Country'] == 2, 'is_Cash_Paid'], label = 'Country 2', sha
sns.kdeplot(df_pd.loc[df_pd['Country'] == 3, 'is_Cash_Paid'], label = 'Country 3', sha
plt.xlabel('is_Cash_Paid')
plt.ylabel('Density')
plt.title('Density Plot of is_Cash_Paid by Directions')
```

Out[42]:
```
Text(0.5, 1.0, 'Density Plot of is_Cash_Paid by Directions')
```



3.4.3 Numerical correlations

In [43]:
```
# Correlations of numerical values
df_pd.corr()['Country'].sort_values()
```

Out[43]:
```
Directions         -0.334543
Stops              -0.122162
Distance           -0.044784
When_Fuel_Added    -0.009797
is_Fuel_Added       0.059644
is_Cash_Paid        0.137658
Country             1.000000
Name: Country, dtype: float64
```

3.5 Generate Training and Testing Data Sets

3.5.1 Select most correlated 3 columns

In [71]:
```
# 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 [74]:
```
X_train, X_test, y_train, y_test = ML_DataSet(df_pd)
```

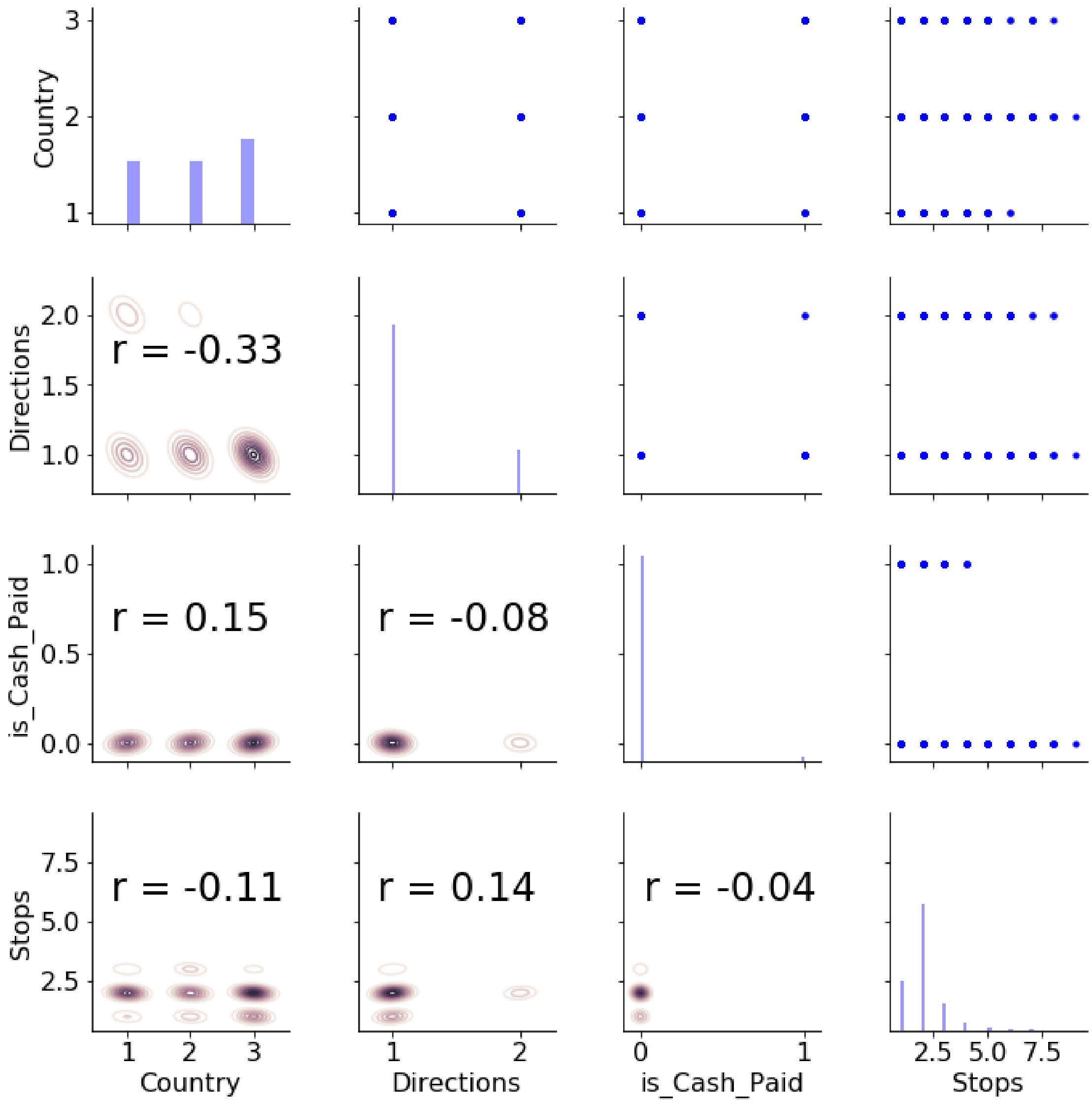
In [75]:
```
X_train.head(5)
```

Out[75]:

	Country	Directions	is_Cash_Paid	Stops
9057	3	1	0	2
9293	3	1	0	3
1721	1	1	0	3
7296	3	1	0	2
5641	2	1	0	2

3.5.2 Plots of Selected Columns Correlation Coefficient

In [76]:
```
# 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)
# 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 [77]:
```
# 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 [78]:
```
# 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 [79]:
```
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: 0.6984
Baseline RMSE: 0.8357
```

4.3 Machine Learning Approaches

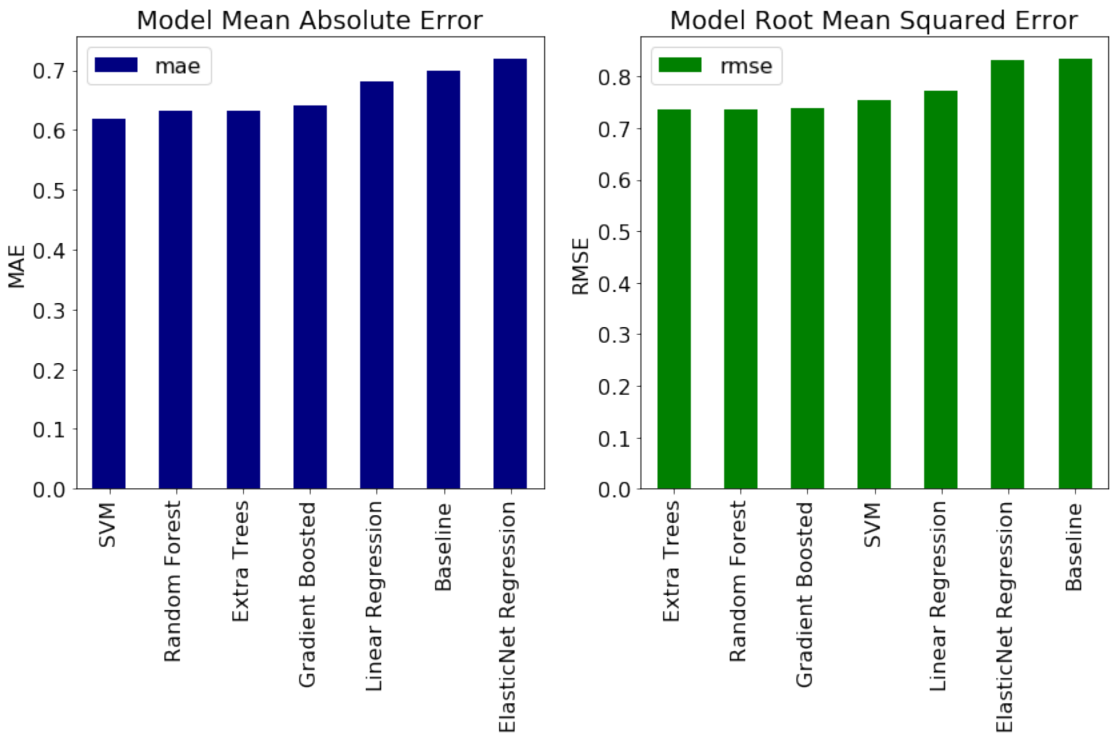
In [80]:
```
# Evaluate machine learning approaches by training on training set and testing on tes
def evaluate(X_train, X_test, y_train, y_test):
    # ML approach list including baseline above
    model_name_list = ['Linear Regression', 'ElasticNet Regression', 'Random Forest',
                       '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 [81]: # run ML approach and evaluation
         results = evaluate(X_train, X_test, y_train, y_test)
```

4.4 Visual Comparison of ML Approaches

```
In [82]: 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 [83]:
```
# show quantitative results
results
```

Out[83]:

	mae	rmse
Linear Regression	0.681267	0.770967
ElasticNet Regression	0.720266	0.830973
Random Forest	0.631355	0.736762
Extra Trees	0.631399	0.736655
SVM	0.618055	0.753042
Gradient Boosted	0.642162	0.738899
Baseline	0.698363	0.835681

In [84]:
```
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 9.60% better on MAE than the baseline.
The Random Forest is 11.84% better on RMSE than the baseline.
```

4.5 Interpretable Formula with Ordinary Linear Regression

In [85]:
```
lr = LinearRegression()
lr.fit(X_train.drop(columns='Country'), y_train)
```

```python
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 = 2.99 + -0.64 * Directions + 0.64 * is_Cash_Paid + -0.06 * 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, a solution is developed to use Spark (with pyspark library) on a Hadoop cluster (a group of multiple servers). The data engineering steps can be easily performed with Spark SQL & Optimus functions, such as select(), where(), groupBy(), cols.append(), cols.drop(), rows.sort() and so on. Moreover, Spark dataframe can be converted into Pandas dataframe with the function .toPandas(). It is very flexible and scalable for data analytics implementation.