CS156_Assignment_5

March 30, 2018

```
In [2]: from google.colab import files
        uploaded = files.upload()
```

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

```
Saving anonymized.csv to anonymized (4).csv
```

1 1. Build a density model for the number of transactions that occur in a month.

```
In [3]: import pandas as pd
        import io
        import datetime
        import numpy as np

        # Read anonymized.csv that is uploaded to Google Colaboratory
        df = pd.read_csv(io.StringIO(uploaded['anonymized.csv'].decode('utf-8')))

        # Converting Date column to python datetime object
        df['Date'] = [month[2:] for month in df.Date]
        df['Date'] = pd.to_datetime(df['Date'])

        # Adding new column Month in the form 'Month Year'
        df['Month'] = df['Date'].apply(lambda x: x.strftime('%B %Y'))

        # Sorting dataframe in ascending order of Months
        df = df.iloc[pd.to_datetime(df.Month).values.argsort()]

        # Create a frequency table for transactions each month
        df['Frequency'] = df.groupby('Month')['Month'].transform('count')

        # Drop duplicate rows in Month, keeping the first entry
        df = df.drop_duplicates(subset='Month', keep='first', inplace=False)

        # Output Frequency as an array
```

```
transactions_in_a_month = df['Frequency']

# Dropping columns Date and Amount
df = df.drop(['Date','Amount'], axis=1)

df.head()
```

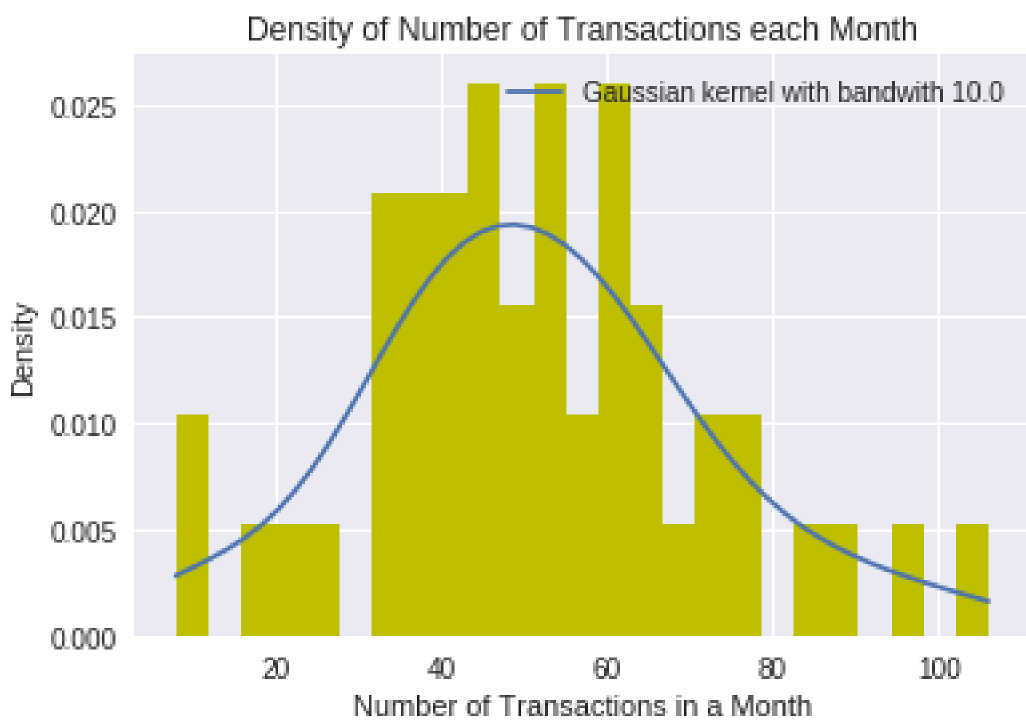
Out[3]:

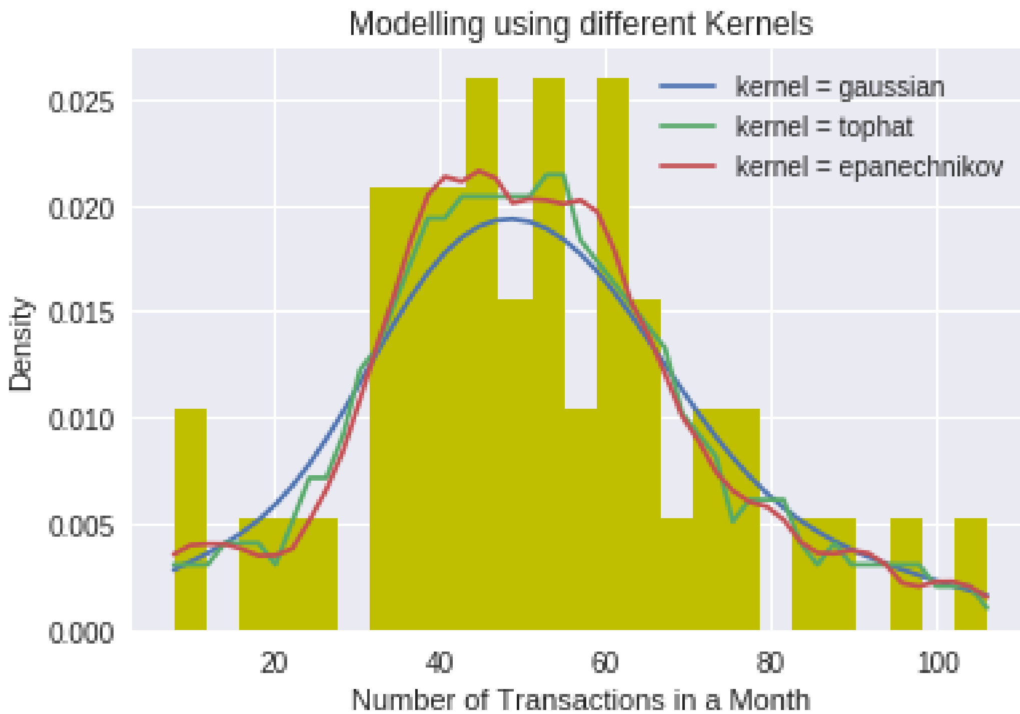
	Month	Frequency
72	September 2013	10
601	October 2013	16
1691	November 2013	8
75	December 2013	27
1916	January 2014	40

In [4]:

```
from sklearn.neighbors.kde import KernelDensity
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV

%matplotlib inline

# Plot histogram for transactions
plt.hist(transactions_in_a_month,normed=True,bins=25,color='y')

xaxis = np.linspace(min(transactions_in_a_month), max(transactions_in_a_month),
                    len(transactions_in_a_month))[:, np.newaxis]

# Finding the optimal bandwith for gaussian kernel using k-fold cross validation
bandwidths = 10 ** np.linspace(-1, 1, 100)
grid = GridSearchCV(KernelDensity(kernel='gaussian'),{'bandwidth': bandwidths},
                    cv=15)
grid.fit(transactions_in_a_month.as_matrix().reshape(-1,1))
best_bandwith = grid.best_params_['bandwidth']

# Initialize gaussian KDE model with optimal bandwith value
# Fit model to data and plot log probability distribution
kde_gaussian = KernelDensity(kernel='gaussian', bandwidth=best_bandwith)
kde_gaussian.fit(transactions_in_a_month.as_matrix().reshape(-1,1))
log_dens_gaussian = kde_gaussian.score_samples(xaxis.reshape(-1,1))

plt.plot(xaxis,np.exp(log_dens_gaussian),label = 'Gaussian kernel with bandwith {}'.form
plt.title("Density of Number of Transactions each Month")
plt.legend(loc='best')
plt.xlabel("Number of Transactions in a Month")
plt.ylabel("Density")
plt.show()

# Visualization for other kernels
plt.hist(transactions_in_a_month,normed=True,bins=25,color='y')
```

```
for kernel in ['gaussian', 'tophat', 'epanechnikov']:
  kde = KernelDensity(kernel=kernel, bandwidth=best_bandwith)
  kde.fit(transactions_in_a_month.as_matrix().reshape(-1,1))
  log_dens = kde.score_samples(xaxis.reshape(-1,1))
  plt.plot(xaxis,np.exp(log_dens), label="kernel = {}".format(kernel))

plt.legend(loc='best')
plt.title("Modelling using different Kernels")
plt.xlabel("Number of Transactions in a Month")
plt.ylabel("Density")
plt.show()
```





2 2. Build a density model for number of transactions on each day of the month.

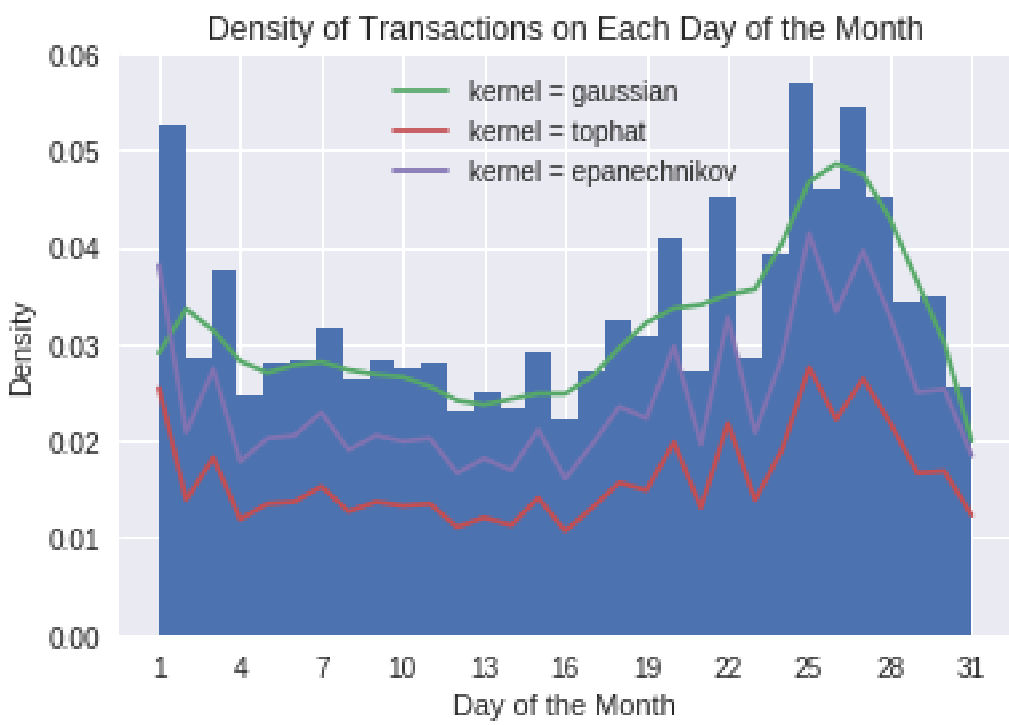
In [5]:
```
import time

# Read anonymized.csv that is uploaded to Google Colaboratory
# into a different dataframe
df1 = pd.read_csv(io.StringIO(uploaded['anonymized.csv'].decode('utf-8')))

# Parse string format in the data into day of the month (1-31)
df1['Day'] = df1.Date.apply(lambda x: time.strptime(x,'%d%b%Y').tm_mday)

# Create frequency table for each day of the month
df1['Frequency'] = df1.groupby('Day')['Day'].transform('count')

# Dropping columns Date
df1 = df1.drop(['Date'], axis=1)

df1.head()
```

Out[5]:

	Amount	Day	Frequency
0	54241.35	25	139
1	54008.83	29	84

```
2  54008.82   30          85
3  52704.37    5          68
4  52704.36   23          70
```

In [6]:
```python
# Plot histogram for transactions
df1.Day.hist(normed=True,bins=31)

days = np.arange(1,32)
xaxis1 = np.linspace(min(df1.Day), max(df1.Day),31)[:, np.newaxis]

for kernel in ['gaussian', 'tophat', 'epanechnikov']:
    # Use bandwith = 1 as default since size of each point varies daily
    kde1 = KernelDensity(kernel=kernel, bandwidth=1)
    kde1.fit(df1.Day.values.reshape(-1,1))
    log_dens_days = kde1.score_samples(xaxis1.reshape(-1,1))

    plt.plot(days,np.exp(log_dens_days), label="kernel = {}".format(kernel))
    plt.xticks(range(1,32,3))

plt.title("Density of Transactions on Each Day of the Month")
plt.legend(loc='best')
plt.xlabel("Day of the Month")
plt.ylabel("Density")
plt.show()
```



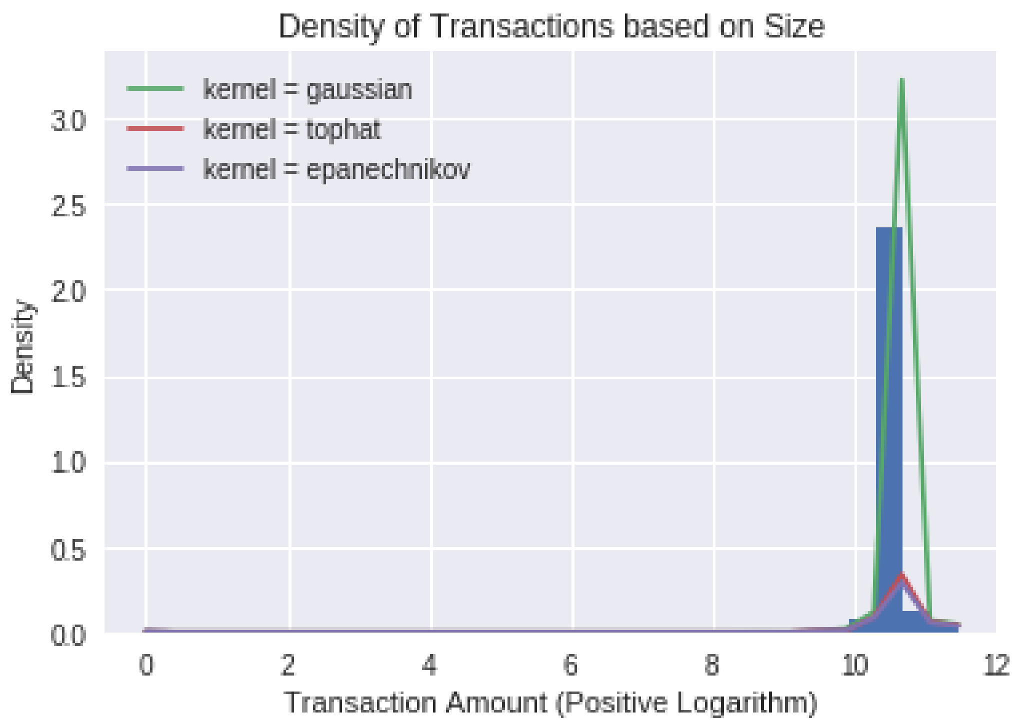
3 3. Build a density model for the transaction size

In [7]:
```
# Read anonymized.csv that is uploaded to Google Colaboratory
# into a different dataframe
df2 = pd.read_csv(io.StringIO(uploaded['anonymized.csv'].decode('utf-8')))

# Transactions spanning several orders of magnitude are best modeled in log-space
# Cannot directly take log since dataset contain negative numbers
# Transform dataset to positive based on the smallest negative value
df2['Log Amount'] = np.log(df2.Amount - min(df2.Amount) + 1)

df2.head()
```

Out[7]:

	Date	Amount	Log Amount
0	25May2016	54241.35	11.464757
1	29May2017	54008.83	11.462314
2	30Jun2017	54008.82	11.462314
3	05Jan2017	52704.37	11.448497
4	23Feb2017	52704.36	11.448497

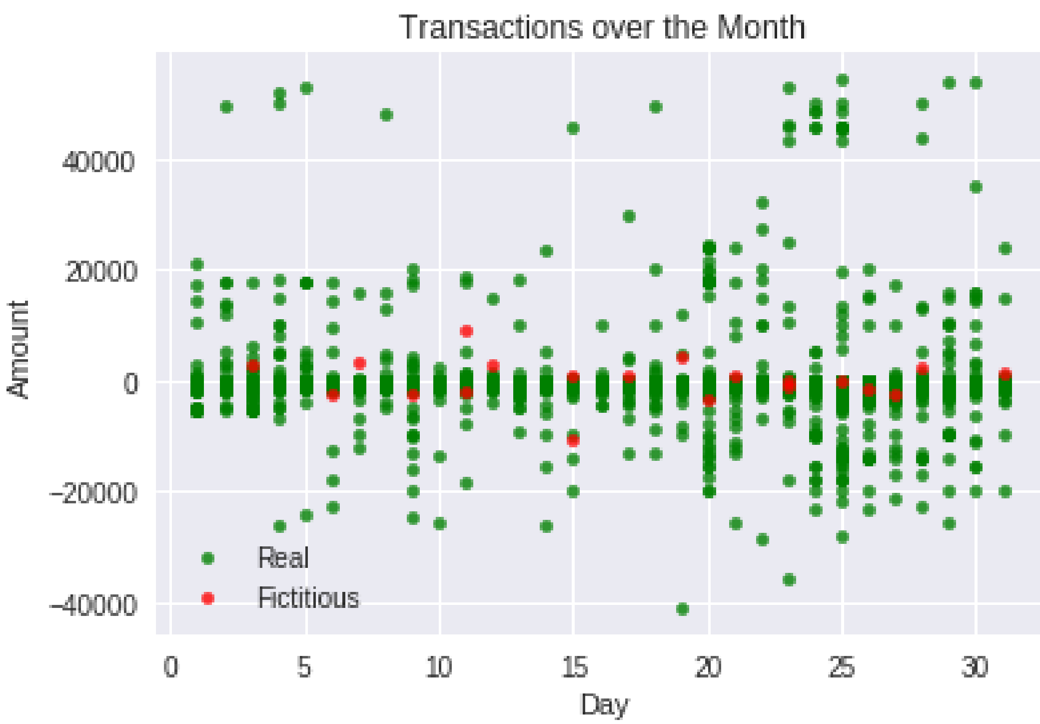
In [8]:
```
log_amount = df2['Log Amount']
log_amount.hist(normed=True, bins=30)

xaxis2 = np.linspace(min(log_amount), max(log_amount),30)[:, np.newaxis]

for kernel in ['gaussian', 'tophat', 'epanechnikov']:
    kde2 = KernelDensity(kernel=kernel, bandwidth=0.05)
    kde2.fit(log_amount.values.reshape(-1,1))
    log_dens_size = kde2.score_samples(xaxis2)

    plt.plot(xaxis2[:,0],np.exp(log_dens_size), label="kernel = {}".format(kernel))

plt.title("Density of Transactions based on Size")
plt.legend(loc='best')
plt.xlabel("Transaction Amount (Positive Logarithm)")
plt.ylabel("Density")
plt.show()
```



4 4. Create a fictitious month of personal transactions.

In [9]:
```
# Instantiate each KDE using Gaussian kernels
kde_month = KernelDensity(kernel='gaussian', bandwidth=10)
kde_month.fit(transactions_in_a_month.as_matrix().reshape(-1,1))

kde_day = KernelDensity(kernel='gaussian', bandwidth=1)
kde_day.fit(df1.Day.values.reshape(-1,1))

kde_amount = KernelDensity(kernel='gaussian', bandwidth=0.05)
kde_amount.fit(log_amount.values.reshape(-1,1))

# Sample number of transactions in a month as an int
no_of_transactions = int(kde_month.sample()[0][0])

days = []      # days that each transaction took place
amounts = []   # amount of each transaction

# Generate a given number of transactions, the transaction day and amount
for i in range(no_of_transactions):
    day_of_transaction = int(kde_day.sample())
    while not 0 < day_of_transaction <= 31:  # days have to be within months
        day_of_transaction = int(kde_day.sample())
```

```python
    days.append(day_of_transaction)
    # Sampled transaction amount is a log amount that needs to be exponentiated
    transaction_amount = (np.exp(kde_amount.sample()[0][0])) + min(df2.Amount) - 1
    amounts.append(transaction_amount)

# Output data as a dataframe rounded off to 2 decimal points
fictitious_month = pd.DataFrame({'Day': days,'Amount': amounts})
fictitious_month = fictitious_month.round(decimals = 2)

# Plot real and fictitious data points
ax1 = df1.plot(x='Day',y='Amount',kind='scatter',color='g',label='Real',alpha=0.8)
fictitious_month.plot(x='Day',y='Amount',kind='scatter',ax=ax1,color='r',
                     label='Fictitious',alpha=0.8)
plt.title("Transactions over the Month")
plt.show()
```



5 5. Explain what flaws still remain in your model that a forensic accountant might be able to find and determine that this was a fraudulent set of transactions.

This model tends to model fictitious transactions after highest density data. In other words, it disguises itself in the data where it is densest. As we can see from the graph, our fictitious data tend to be centred around a large density of real data, near the origin. Our model has a good idea of the number of transactions that occur each month, the days where more transactions take place, and the typical transaction size in order to generate data with a realistic number of transactions (40-60 transactions a month), days that most transactions occur (towards the end of the month), and at the size that is most often (a small amount above and below zero). To the extent that they are realistic and do not draw attention to them as an outlier, our model is a "good" set of fictitious data.

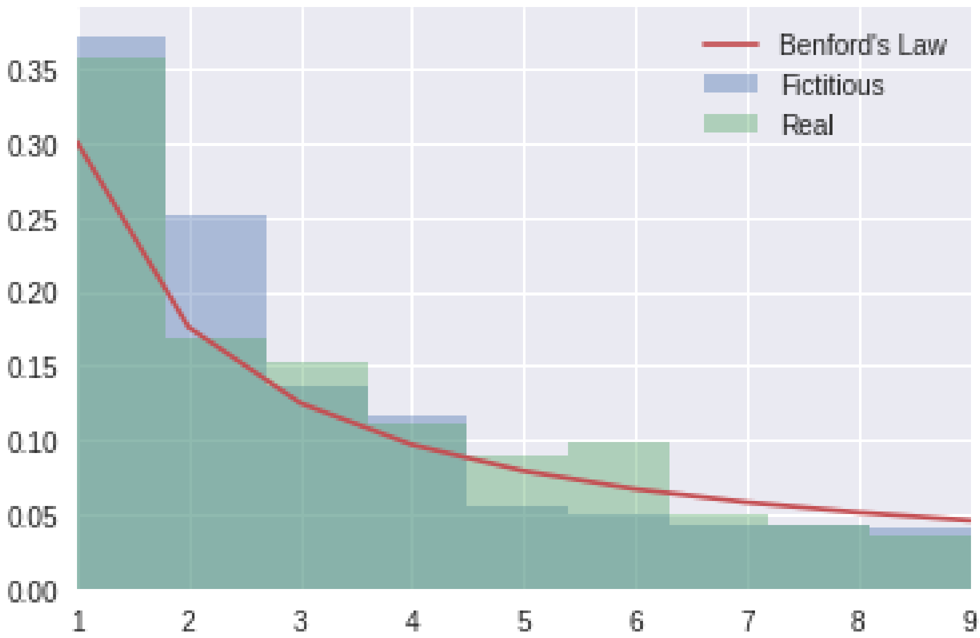
Nevertheless, there are real-world information that this model do not encompass. For instance, other transactional information such as location of transaction, individual spending pattern and items purchased could be used during forensic analysis to discover anomalies in the set of transactions. Since the model does not know these dimensions of the data, it might generate datapoints that are not realistic and can be immediately obvious to a forensic accountant, such as an expenditure entry of $50,000 at a grocery store. To overcome this, we need to identify and include other dimensions of the dataset that can influence how a transaction takes place. One can also include feature engineering methods such as Principal Component Analysis or Linear Disciminant Analysis to find components that have stronger biases than others, and weight them accordingly.

At an eye's glance, our dataset which includes real and made-up data appear healthy with regression to the mean, and a relatively small number of outliers. This also makes it hard for the forensic accountant, who tends to nitpick the outliers first. This model also included approximately 2,800 real datapoints and a much smaller number of fake datapoints. Given a large dataset, the relatively small probability of finding a fraudulent transaction work against the forensic accountant. If the forensic accountant samples a data at random, there is a higher likelihood that the transaction is authentic than fictitious. This however, becomes easier for the accountant if the fraudster becomes greedy and generate too many fraudulent transactions. Similarly, if the fraudster attempts to get rish by forging a large transaction amount, it is also immediately obvious to the forensic accountant since most transactions are small and negative.

6 6. How well does the data follow Benford's law?

In [10]:
```
benford_numbers = [np.log10(i + 1) - np.log10(i) for i in range (1,10)]
digits = range(1,10)

kde_month_ = KernelDensity(kernel='gaussian', bandwidth=10)
kde_month_.fit(transactions_in_a_month.as_matrix().reshape(-1,1))

kde_day_ = KernelDensity(kernel='gaussian', bandwidth=1)
kde_day_.fit(df1.Day.values.reshape(-1,1))

kde_amount_ = KernelDensity(kernel='gaussian', bandwidth=0.05)
kde_amount_.fit(log_amount.values.reshape(-1,1))

days_ = []
amounts_ = []

# Generate equal amount of fictitious data to real data
for i in range(len(df1.Amount)):
    day_of_transaction_ = int(kde_day_.sample())
```

```
        while not 0 < day_of_transaction_ <= 31: # days have to be within months
            day_of_transaction_ = int(kde_day_.sample())

        days_.append(day_of_transaction_)

        transaction_amount_ = (np.exp(kde_amount_.sample()[0][0])) + min(df2.Amount) - 1
        amounts_.append(transaction_amount_)

    fictitious_month_ = pd.DataFrame({'Day': days_,'Amount': amounts_})
    fictitious_month_ = fictitious_month_.round(decimals = 2)

    # Turning fictitious and real data into arrays
    fictitious_data = [int(str(abs(amount))[0]) for amount in fictitious_month_.Amount]
    real_data = [int(str(abs(amount))[0]) for amount in df1.Amount]

    # Plot histograms and Benford's Law
    fictitious_points = plt.hist(fictitious_data,normed=True,label='Fictitious',alpha=0.4,r
    real_points = plt.hist(real_data,normed=True,label='Real',alpha=0.4,rwidth=1)
    plt.plot(digits, benford_numbers,label='Benford\'s Law')
    plt.legend(loc='best')
    plt.xlim(1,9)
    plt.show()
```




```
In [11]: from sklearn.metrics import mean_squared_error

         # Mean Squared Error of green histogram to Benford's Law
```

```
mse_real = mean_squared_error(list(real_points[0][1:]), benford_numbers)
print mse_real

# Mean Squared Error of blue histogram to Benford's Law
mse_fictitious = mean_squared_error(list(fictitious_points[0][1:]), benford_numbers)
print mse_fictitious
```

```
0.0006212786285946714
0.001381348578404312
```

The fictitious data follows Benford's Law to a smaller extent than the real data. The real and fictitious histograms are generated from the same amount of fictitious and real date, however, two evidence supports this argument. Firstly, on the fictitious (blue) histogram, the leading digit 5 has near-zero density and therefore deviates from Benford's Law. Secondly, the mean squared error of this fictitious data with Benford's Law is 0.0013, twice that of the real data which is 0.0006. Assuming the mean squared error as the cost function, the accuracy of our fictitious data is 99.87% while the real data produced a score of 99.94%.