Log Revenue data

Bruna Wundervald

October, 2018

```
# Packages and models import
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.ensemble import RandomForestRegressor
```

```
## /Users/brunawundervald/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/weight_boosting.py:29:
##   from numpy.core.umath_tests import inner1d
```

```
from sklearn.metrics import mean_squared_error, r2_score
import csv
import pandas as pd
import numpy as np
import json

import random
import seaborn as sns
from pandas.io.json import json_normalize
import matplotlib.pyplot as plt
```

```
# Reading & manipulating data

# Step 1: deal with JSON data

# Columns that have json format
columns = ['device', 'geoNetwork', 'totals', 'trafficSource']

def json_read():
    data_frame =  'data/all/train.csv'

    #Importing the dataset
    df = pd.read_csv(data_frame,
    # loading the json columns properly
                     converters={column: json.loads for column in columns}, # transforming this column
                     dtype={'fullVisitorId': 'str'},
                     skiprows=lambda i: i>0 and random.random() > 0.5)

    for column in columns:
    # loop to finally transform the columns in data frame
        #It will normalize and set the json to a table
        column_as_df = json_normalize(df[column])
        # here will be set the name using the category
        # and subcategory of json columns
        column_as_df.columns = [f"{column}.{subcolumn}" for subcolumn in column_as_df.columns]
        # after extracting the values, let drop the original columns
        df = df.drop(column, axis=1).merge(column_as_df, right_index=True, left_index=True)

    # returning the df after importing and transforming
    return df
```

```
da = json_read()
samp = da.sample(10000)
```

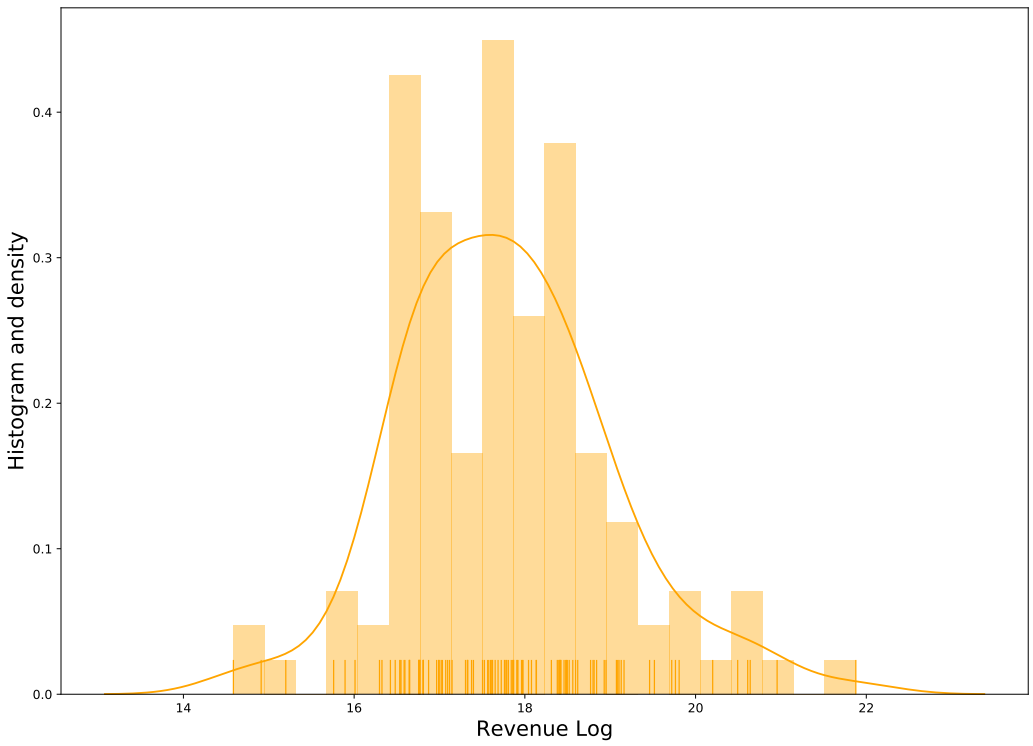
```
# Dependent variable plot
samp['totals.transactionRevenue'] = samp['totals.transactionRevenue'].astype(float)
log_revenue = np.log(samp[samp['totals.transactionRevenue'] > 0]["totals.transactionRevenue"] + 0.01)

plt.clf() # clean plot environment
plt.figure(figsize = (14, 10))
```

```
sns.distplot(log_revenue, bins = 20, kde = True, color = 'orange', rug = True)
```

```
plt.xlabel('Revenue Log', fontsize = 17)
```

```
plt.ylabel('Histogram and density', fontsize = 17)
```

```
plt.show() # show plot
```




```
# Selecting predictor variables!

# Dimensions of sample
samp.shape
```

```
samp.fillna(0, inplace = True)

# Selecting the independent variables ----------------------------------
```

```
X = samp[['socialEngagementType', 'device.isMobile', 'device.browser', 'totals.pageviews', 'totals.hits
y = np.log(samp[['totals.transactionRevenue']] + 0.01)

# Defining function to convert variables in dummies
def dummy(var):
    X[var] = X[var].astype('category')
    X[var] = X[var].cat.codes
    return X

X = dummy('socialEngagementType')
```

```
X = dummy('device.browser')
X = dummy('geoNetwork.country')
X = dummy('trafficSource.source')
X = dummy('device.operatingSystem')
X = dummy('trafficSource.adwordsClickInfo.adNetworkType')

X.iloc[0] # All ok

# Train and test split (automatic function)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4)

# Setting the model(s)
model_lm = linear_model.LinearRegression()
model_lm.fit(X_train, y_train)


# Model coefficients
```

```
model_lm.coef_
```

```
model_rf = RandomForestRegressor()
model_rf.fit(X_train, y_train)



# Predictions
```

```
y_pred_lm = model_lm.predict(X_test)
y_pred_rf = model_rf.predict(X_test)
```

```
print("LM - Mean squared error: %.2f"
      % mean_squared_error(y_test, y_pred_lm))
```

```
## LM - Mean squared error: 4.15
```

```
print("LM - Mean squared error: %.2f"
      % mean_squared_error(y_test, y_pred_rf))
```

```
## LM - Mean squared error: 4.75
```