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
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression


import csv
import pandas as pd
import numpy as np
import json
from pandas.api.types import is_object_dtype
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/rstudio/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()
# da.to_csv("logrevenue.csv", sep='\t')

# Using a sample to make it faster
samp = da.sample(10000)
```

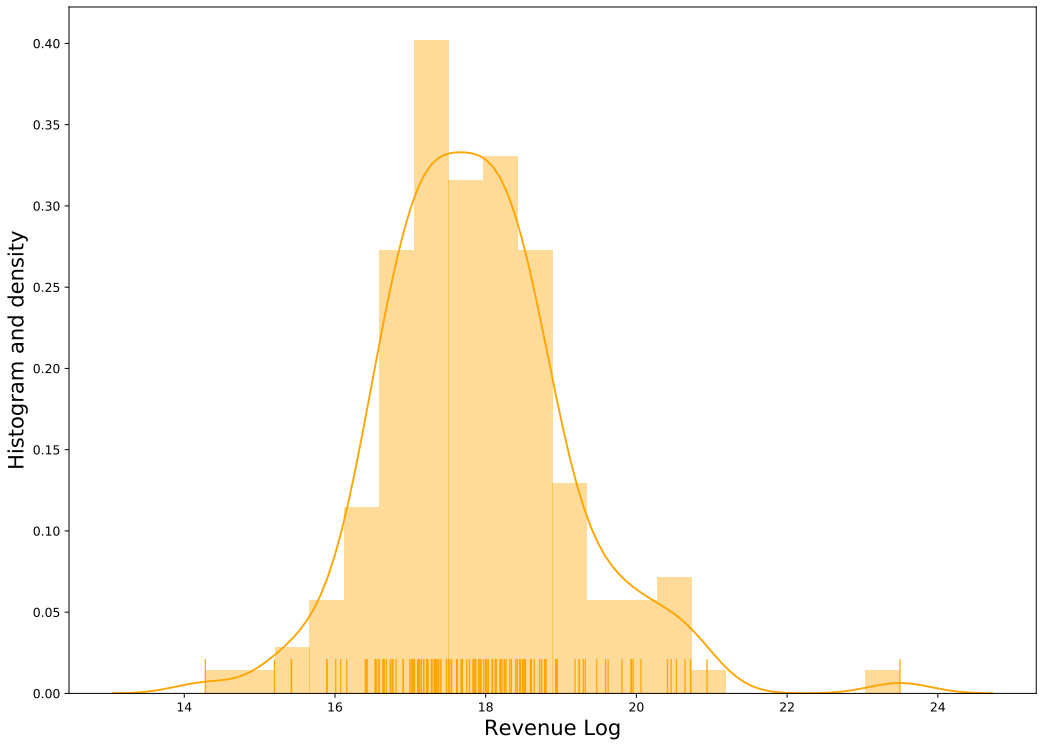
```
# Response variable plot
samp['totals.transactionRevenue'] = samp['totals.transactionRevenue'].astype(float)

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
```



```
# Dimensions of sample
samp.shape

# Filling NaNs
```

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

# Selecting the covariables ----------------------------------
# X = samp[['socialEngagementType', 'device.isMobile', 'device.browser', 'totals.pageviews', 'totals.hi

y = np.log(samp[['totals.transactionRevenue']] + 0.01)


X = samp.drop(['totals.transactionRevenue'], axis = 1)

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


columns = X.columns

for index in range(0, len(columns)):
  if is_object_dtype(X[columns[index]]) == True:
    X = dummy(columns[index])
  else:
    X = X

X.iloc[0] # All ok
```

```
X.shape

# Selecting the 10 best features based on univariate statistical tests
```

```
X_new = SelectKBest(f_regression, k = 10).fit_transform(X, y)

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

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

# Setting the model(s)
# ------------------------------
# 1. Linear Regression
# 2. Random Forests
# 3. Neural Networks
# --------------------------------

# 1. Linear Regression
model_lm = linear_model.LinearRegression()
model_lm.fit(X_train, y_train)

# 2. Random Forests
```

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

# 3. Neural Networks
# Scaling data for the NN
```

```
scaler = StandardScaler()
scaler.fit(X_train)
```

```
X_train_nn = scaler.transform(X_train)
```

```
X_test_nn = scaler.transform(X_test)
```

```
model_nn = MLPRegressor()
model_nn.fit(X_train_nn, y_train)

# Predictions
```

```
y_pred_lm = model_lm.predict(X_test)
y_pred_rf = model_rf.predict(X_test)
y_pred_nn = model_nn.predict(X_test_nn)
```

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

```
## LM - Mean squared error: 7.80
```

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

```
## RF - Mean squared error: 7.30
```

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

```
## NN - Mean squared error: 7.52
```

Some conclusions

The linear regression and neural network models are predicting better than the rnadom forests when we use the 10 best predictor variables. Even so, we should always consider that linear regression requires less computational effort than that of the neural networks, so, in the case their predictions are similar, we can opt for the first one.