kickstarter database.ipynb

파일 수정 보기 삽입 런타임 도구 도움말 10월 22일에 마지막으로 수정됨

댓글 공유

+ 코드 + 텍스트 연결 수정 가능

```
import os
# Find the latest version of spark 2.0  from http://www-us.apache.org/dist/spark/ and enter as the spark version
# For example:
# spark_version = 'spark-2.4.6'
spark_version = 'spark-2.4.7'
os.environ['SPARK_VERSION']=spark_version

# Install Spark and Java
!apt-get update
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q http://www-us.apache.org/dist/spark/$SPARK_VERSION/$SPARK_VERSION-bin-hadoop2.7.tgz
!tar xf $SPARK_VERSION-bin-hadoop2.7.tgz
!pip install -q findspark

# Set Environment Variables
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = f"/content/{spark_version}-bin-hadoop2.7"

# Start a SparkSession
import findspark
findspark.init()
```

```
Get:1 https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/ InRelease [3,626 B]
Ign:2 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  InRelease
Ign:3 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64  InRelease
Get:4 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  Release [697 B]
Hit:5 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64  Release
Get:6 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  Release.gpg [836 B]
Hit:7 http://archive.ubuntu.com/ubuntu bionic InRelease
Get:8 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic InRelease [15.9 kB]
Get:9 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:10 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Get:11 https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/ Packages [39.1 kB]
Hit:13 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu bionic InRelease
Ign:14 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  Packages
Get:15 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:14 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  Packages [334 kB]
Get:16 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic/main Sources [1,681 kB]
Get:17 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [2,110 kB]
Get:18 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [1,341 kB]
Get:19 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages [2,104 kB]
Get:20 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [1,693 kB]
Get:21 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic/main amd64 Packages [860 kB]
Fetched 10.4 MB in 4s (2,739 kB/s)
Reading package lists... Done
```

[ ]
```
!wget https://jdbc.postgresql.org/download/postgresql-42.2.9.jar
```

```
--2020-10-11 13:42:33--  https://jdbc.postgresql.org/download/postgresql-42.2.9.jar
Resolving jdbc.postgresql.org (jdbc.postgresql.org)... 72.32.157.228, 2001:4800:3e1:1::228
Connecting to jdbc.postgresql.org (jdbc.postgresql.org)|72.32.157.228|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 914037 (893K) [application/java-archive]
Saving to: ‘postgresql-42.2.9.jar’

postgresql-42.2.9.j 100%[===================>] 892.61K  3.55MB/s    in 0.2s

2020-10-11 13:42:34 (3.55 MB/s) - ‘postgresql-42.2.9.jar’ saved [914037/914037]
```

[ ]
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("CloudETL").config("spark.driver.extraClassPath","/content/postgresql-42.2.9.jar").getOrCreate()
```

[ ]
```
# Read in data from S3 Buckets
from pyspark import SparkFiles
url ="https://laurentvh-kickstarter.s3.us-east-2.amazonaws.com/latest_data.csv"
spark.sparkContext.addFile(url)
kick_df = spark.read.csv(SparkFiles.get("latest_data.csv"), sep=",", header=True, inferSchema=True)

# Show DataFrame
kick_df.show()
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-4-916a9ce8aa55> in <module>()
      3 url ="https://laurentvh-kickstarter.s3.us-east-2.amazonaws.com/latest_data.csv"
      4 spark.sparkContext.addFile(url)
----> 5 kick_df = spark.read.csv(SparkFiles.get("latest_data.csv"), sep=",", header=True, inferSchema=True)
      6
      7 # Show DataFrame

                              4 frames

/usr/lib/python3.6/socket.py in readinto(self, b)
    584         while True:
    585             try:
--> 586                 return self._sock.recv_into(b)
    587             except timeout:
    588                 self._timeout_occurred = True
```

```
KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

```
kick_df.dtypes
```

```
# Configure settings for RDS

mode = "append"
jdbc_url="jdbc:postgresql://kickstarter.c90yn2pvfvlh.us-east-2.rds.amazonaws.com:5432/postgres"
config = {"user":"postgres",
          "password": "Laurent123!",
          "driver":"org.postgresql.Driver"}
```

```
# Write DataFrame to mask table in RDS
kick_df.write.jdbc(url=jdbc_url, table='kickstarter', mode=mode, properties=config)
```

```
# Import our dependencies
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder
import pandas as pd
import tensorflow as tf
import numpy as np
```

```
# Change the PySpark dataframes into Pandas dataframes
kick_df = kick_df.select("*").toPandas()
```

Preprocessing for machine learning

We will inspect the data to see if there's any categorical variable or NA values that we need to drop.

```
# Generate our categorical variable list
df_cat = df.dtypes[df.dtypes == "object"].index.tolist()
```

```
kick_df_cat = kick_df.dtypes[kick_df.dtypes == "object"].index.tolist()
```

I. Inspect whether we need bucketing of variables in categorical columns

```
df[df_cat].nunique()
```

```
kick_df[kick_df_cat].nunique()
```

II. Encode the categorical variables

First, we will inspect each column on NA values and whether we need to bucket any of the values together in each categorical column. Bucketing them if needed

```
# Print out each Category value counts of a categorical column
cate_counts = df.ColumnName.value_counts()

# Visualize the value counts
cate_counts.plot.density()

# Determine which values to replace
replace = list(cate_counts[cate_counts < #].index)

# Replace in DataFrame
for value in replace:
    df.ColumnName = df.ColumnName.replace(value, "Bucket")
# Check to make sure binning was successful
df.ColumnName.value_counts()
```

Then, move onto encoding the categorical columns. OR just skip to this part if bucketing is unncessary.

```
# Create a OneHotEncoder instance
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder(sparse=False)

# Fit and transform the OneHotEncoder using the categorical variable list
encode_df = pd.DataFrame(enc.fit_transform(df_ColumnName.values.resape(-1,1)))

# Add the encoded variable names to the DataFrame
encode_df.columns = enc.get_feature_names(['ColumnName'])
```

```
encode_df.head()
```

Finally, merge the encoded DataFrame with the original df and drop the original columns.

```
df.merge(encode_df, left_index=True, right_index=True).drop("ColumnName", 1)
```

III. Decide on columns to drop

Inspect null values in each column and decide whether it's worth to drop or fill NA with 0s if needed. Then, we will drop columns that are not adding valuable information such as "name" and id" columns in the kick_df.

IV. Trial and Error for Machine Learning models: Random Forests versus Neural Networks

Since we will be developing a model that can identify whether a project will success with the funding, we will be using a binary classification.

To get started we will define features and the output.

```
# Split our preprocessed data into our features and target arrays
y = new_df["OutputColumn"].values
X = new_df.drop(["OutputColumn"],1).values

For our dataset, the output column will be "state" column.
# Split the data into testing and training dataset before standardizing the data.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=78)
```

VI. Standardize the data

```
# Create a StandardScaler instance
scaler = StandardScaler()

# Fit the StandardScaler
X_scaler = scaler.fit(X_train)

# Scale the data
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

VII. Random Forests

```
# Create a random forest classifier.
rf_model = RandomForestClassifier(n_estimators=128, random_state=78)

# Fitting the model
rf_model = rf_model.fit(X_train_scaled, y_train)

# Evaluate the model
y_pred = rf_model.predict(X_test_scaled)
print(f" Random forest predictive accuracy: {accuracy_score(y_test,y_pred):.3f}")
```

VIII. Support Vector Machine

```
# Create the SVM model
svm = SVC(kernel='sigmoid')
# Train the model
svm.fit(X_train, y_train)
# Evaluate the model
y_pred= svm.predict(X_test_scaled)
print(f" SVM model accuracy: {accuracy_score(y_test,y_pred):.3f}")
```

IX. Neural Networks

```
# Define the model - deep neural net
number_input_features = len(X_train[0])
hidden_nodes_layer1 =  8 # number of neurons will change depending on the nature of the dataset (2-3 times the number of
hidden_nodes_layer2 = 5

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(
```

```
    tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation="relu")
)

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="relu"))

# Output layer
nn.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Check the structure of the model
nn.summary()

# Compile the model
nn.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])

# Train the model
fit_model = nn.fit(X_train,y_train,epochs=100)

# Evaluate the model using the test data
model_loss, model_accuracy = nn.evaluate(X_test,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
[ ] # Creating a dataframe grouped by category with columns for failed and successful
    cat_df = pd.get_dummies(df.set_index('category').state).groupby('category').sum()

    # Plotting
    fig, ((ax1, ax2), (ax3, ax4), (ax5, ax6)) = plt.subplots(3, 2, figsize=(12,12))

    color = cm.CMRmap(np.linspace(0.1,0.8,df.category.nunique())) # Setting a colormap

    df.groupby('category').category.count().plot(kind='bar', ax=ax1, color=color)
    ax1.set_title('Number of projects')
    ax1.set_xlabel('')

    df.groupby('category').usd_goal.median().plot(kind='bar', ax=ax2, color=color)
    ax2.set_title('Median project goal ($)')
    ax2.set_xlabel('')

    df.groupby('category').usd_pledged.median().plot(kind='bar', ax=ax3, color=color)
    ax3.set_title('Median pledged per project ($)')
    ax3.set_xlabel('')

    cat_df.div(cat_df.sum(axis=1), axis=0).successful.plot(kind='bar', ax=ax4, color=color) # Normalizes counts across rows
    ax4.set_title('Proportion of successful projects')
    ax4.set_xlabel('')

    df.groupby('category').backers_count.median().plot(kind='bar', ax=ax5, color=color)
    ax5.set_title('Median backers per project')
    ax5.set_xlabel('')

    df.groupby('category').pledge_per_backer.median().plot(kind='bar', ax=ax6, color=color)
    ax6.set_title('Median pledged per backer ($)')
    ax6.set_xlabel('')

    fig.subplots_adjust(hspace=0.6)
    plt.show()
```