Big_Data_Technical_Project_Using_Spark_DataBricks_Kowsik_B

March 7, 2021

[ ]:
```
print('Welcome to Big Data Analytics - Technical Project - Walmart Sales␣
 ↪Analytics Prediction')

############################################# Kowsik Bhattacharjee␣
 ↪##################################
############################################# Big Data Analytics Technical␣
 ↪Project #################
################# Walmart Sales Data Analysis and Model Preparation using Big␣
 ↪Data Spark ############

#Big Data Analytics - Technical Report
#Author - Kowsik Bhattacharjee : MSc Big Data analytics
#Title - Walmart Store Sales Forecasting
#Lecturer - Shagufta Henna

#This lab aims to utilize spark for Big Data Analytics Technical Project.

#The lab primarily focuses to i) data analysis ii) preparing datasets to␣
 ↪analyze, and feed to machine learning #models.

#This lab requires to analyze three datasets under the given sections.

#Used DataSets are:

#1) features.csv
#2) stores.csv
#3) train.csv
##########################################################################################
##########################################################################################
```

[ ]:
```
# Import standard Spark Libraries
import pyspark
from pyspark.sql import SQLContext
from pyspark import SparkContext
from pyspark.sql import SQLContext
```

```
from pyspark.sql import SparkSession
from pyspark.sql.types import *
import pyspark.sql.functions as F
from pyspark import SparkFiles
from pyspark.sql.types import StringType, IntegerType, DoubleType, StructField,␣
↪StructType, ArrayType, MapType
```

[ ]:
```
# Schema define in spark for Stores dataset
# Store,Type,Size

schema = StructType([
        StructField("Store", IntegerType(), True),
        StructField("Type", StringType(), True),
        StructField("Size", IntegerType(), True),
         ])
```

[ ]:
```
############# Reading the file stores.csv with spark.read function␣
↪##############

stores_sdf = spark.read.format("csv").option("header", "true").schema(schema).
↪load("/FileStore/shared_uploads/reach2kowsik@gmail.com/stores.csv")

# Print Scehamof Stores data set
stores_sdf.printSchema()

# Showing top 5 records
stores_sdf.show(5)
```

[ ]:
```
stores_sdf.columns
```

[ ]:
```
# Schema define in spark for features dataset
#␣
↪Store,        Date,        Temperature,        Fuel_Price,        MarkDown1,        MarkDow
#␣
↪MarkDown3,        MarkDown4,        MarkDown5,        CPI,        Unemployment,        IsHo

schema = StructType([
        StructField("Store", IntegerType(), True),
        StructField("Date", StringType(), True),
        StructField("Temperature", DoubleType(), True),
        StructField("Fuel_Price", FloatType(), True),
        StructField("MarkDown1", StringType(), True),
        StructField("MarkDown2", StringType(), True),
        StructField("MarkDown3", StringType(), True),
        StructField("MarkDown4", StringType(), True),
        StructField("MarkDown5", StringType(), True),
        StructField("CPI", FloatType(), True),
        StructField("Unemployment", FloatType(), True),
```

```
        StructField("IsHoliday", StringType(), True),
         ])
```

[ ]:
```
############# Reading the file features.csv with spark.read function␣
 ↪##############
#features_sdf = spark.read.csv('/FileStore/shared_uploads/reach2kowsik@gmail.
 ↪com/features.csv', inferSchema=True, header=True)

features_sdf = spark.read.format("csv").option("header", "true").schema(schema).
 ↪load("/FileStore/shared_uploads/reach2kowsik@gmail.com/features.csv")

# Print Sceham of Features data set
features_sdf.printSchema()

# Showing top 5 records
features_sdf.show(5)

#df1 = spark.read.format("csv").load("dbfs:/FileStore/shared_uploads/
 ↪reach2kowsik@gmail.com/stores.csv")
#df2 = spark.read.format("csv").load("dbfs:/FileStore/shared_uploads/
 ↪reach2kowsik@gmail.com/features.csv")
#df3 = spark.read.format("csv").load("dbfs:/FileStore/shared_uploads/
 ↪reach2kowsik@gmail.com/train.csv")
```

[ ]:
```
features_sdf.columns
```

[ ]:
```
# Schema define in spark for train dataset
# Store,Dept,Date,Weekly_Sales,IsHoliday

schema = StructType([
        StructField("Store", IntegerType(), True),
        StructField("Dept", StringType(), True),
        StructField("Date", StringType(), True),
        StructField("Weekly_Sales", FloatType(), True),
        StructField("IsHoliday", StringType(), True),
         ])
```

[ ]:
```
############# Reading the file train.csv with spark.read function␣
 ↪##############
#train_sdf = spark.read.csv('/FileStore/shared_uploads/reach2kowsik@gmail.com/
 ↪train.csv', inferSchema=True, header=True)

train_sdf = spark.read.format("csv").option("header", "true").schema(schema).
 ↪load("/FileStore/shared_uploads/reach2kowsik@gmail.com/train.csv")

# Print Sceham of Train data set
train_sdf.printSchema()
```

```
# Showing top 5 records
train_sdf.show(5)

#df1 = spark.read.format("csv").load("dbfs:/FileStore/shared_uploads/
↪reach2kowsik@gmail.com/stores.csv")
#df2 = spark.read.format("csv").load("dbfs:/FileStore/shared_uploads/
↪reach2kowsik@gmail.com/features.csv")
#df3 = spark.read.format("csv").load("dbfs:/FileStore/shared_uploads/
↪reach2kowsik@gmail.com/train.csv")
```

[ ]:
```
train_sdf.columns
```

[ ]:
```
##### Data Cleaning Activity on Spark Data Frame #####

#1. Drop duplicate rows
#2. Drop Null rows
#3. Instead Dropping rows that have values as 'NaN' or 'NA' will replace any␣
↪empty cells with 0 instead.

#4. Drop columns those are not neccessary for the rest of the section.

#5. Rename Columns as necessary

#Save the result to final_walmart_df data frame

########################

print('Data Cleaning Activity')
```

[ ]:
```
# Drop duplicates from all 3 data frames
train_sdf = train_sdf.dropDuplicates()
features_sdf = features_sdf.dropDuplicates()
stores_sdf = stores_sdf.dropDuplicates()
```

[ ]:
```
# Rename Column in train_sdf
train_sdf_r = train_sdf.withColumnRenamed('Date','Tr_Date').
↪withColumnRenamed('IsHoliday','Tr_IsHoliday').withColumnRenamed('Store',␣
↪'Tr_Store')
print(train_sdf_r.columns)
train_sdf_r.show(5, truncate = False)
```

[ ]:
```
# Rename Column in features_sdf
features_sdf_r = features_sdf.withColumnRenamed('Date', 'Fr_Date').
↪withColumnRenamed('IsHoliday', 'Fr_IsHoliday').withColumnRenamed('Store',␣
↪'Fr_Store')
print(features_sdf_r.columns)
features_sdf_r.show(5, truncate = False)
```

[ ]:
```
# Performing Merge operation between 3 data frames using SQL

features_sdf.createOrReplaceTempView("Feature")
stores_sdf.createOrReplaceTempView("Store")
train_sdf.createOrReplaceTempView("Train")

# Right outer Join between Feature and Train data frame
#f_t_sdf = spark.sql("""SELECT Tr_Store, Dept, Tr_Date, Weekly_Sales,␣
↪Tr_IsHoliday, Fr_Store, Fr_Date, Temperature, Fuel_Price, MarkDown1,␣
↪MarkDown2, MarkDown3, MarkDown4, MarkDown5, CPI, Unemployment, Fr_IsHoliday
#FROM Feature f LEFT JOIN Train t ON t.Tr_Store = f.Fr_Store""")

f_t_sdf = spark.sql("""SELECT f.Store, t.Dept, f.Date, Weekly_Sales,␣
↪Temperature, Fuel_Price, MarkDown1, MarkDown2, MarkDown3, MarkDown4,␣
↪MarkDown5, CPI, Unemployment, f.IsHoliday
FROM Train t RIGHT JOIN Feature f ON t.Store = f.Store and t.IsHoliday = f.
↪IsHoliday""")

f_t_sdf.show(10)
```

[ ]:
```
display(f_t_sdf)
```

[ ]:
```
# making previously merged data frame as temp view for final merge with Store␣
↪data frame

f_t_sdf.createOrReplaceTempView("mer_fet_tra")
```

[ ]:
```
# Join between Merged data frame and Store data frame

f_t_s_sdf = spark.sql("""SELECT s.Store, Type, Size, Dept, Date, Weekly_Sales,␣
↪Temperature, Fuel_Price, MarkDown1, MarkDown2, MarkDown3, MarkDown4,␣
↪MarkDown5, CPI, Unemployment, IsHoliday
FROM mer_fet_tra m INNER JOIN Store s ON s.Store = m.Store""")

f_t_s_sdf.show(10)
```

[ ]:
```
display(f_t_s_sdf)
```

[ ]:
```
drop_list_sdf = ['Date', 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4',␣
↪'MarkDown5', 'Temperature', 'Fuel_Price', 'CPI', 'Unemployment', 'Type',␣
↪'Size']

final_walmart_sdf = f_t_s_sdf.select([column for column in f_t_s_sdf.columns if␣
↪column not in drop_list_sdf])

display(final_walmart_sdf)
```

[ ]:
```
from pyspark.sql.functions import *
```

```
# replacing string to boolean for further Model processing
# Converting Categorical Variable 'IsHoliday' into Numerical Variables.

final_walmart_sdf = final_walmart_sdf.withColumn('IsHoliday',␣
↪translate('IsHoliday', 'TRUE', '1'))
final_walmart_sdf = final_walmart_sdf.withColumn('IsHoliday',␣
↪translate('IsHoliday', 'FALSE', '0'))
print(final_walmart_sdf.columns)
```

[ ]:
```
final_walmart_sdf.dropna()
final_walmart_sdf.fillna(0)
display(final_walmart_sdf)
```

[ ]:
```
from pyspark.ml.feature import StringIndexer, VectorAssembler
```

[ ]:
```
# Now, for the Spark ML, we need to create a feature column that has all␣
↪features concatenated and a single column for labels.

#You can use VectorAssembler() to create a feature vector from all categorical␣
↪and numerical features. Let us call #the call the final vector as features.

#Now, list all columns in the data and store it in a list named 'all_columns'
print('create a feature column and a single column for labels')
```

[ ]:
```
all_columns = final_walmart_sdf.columns # Task
all_columns
```

[ ]:
```
# Now create a list of columns which you don't wan't to include in your␣
↪features, i.e., the labels
drop_columns = ['Store','label']

drop_columns
```

[ ]:
```
columns_to_use = [i for i in all_columns if i not in drop_columns]
columns_to_use
```

[ ]:
```
# create a VectorAssembler object with columns you want to use for the ML␣
↪models. Let us Name the output column as 'features'. These are the features␣
↪that you will use later. Let us name the vector assembler object 'assembler'

print('create a VectorAssembler object')
```

[ ]:
```
assembler = VectorAssembler(inputCols=columns_to_use,outputCol='features')
print('stat_assembler', (str(assembler.params), columns_to_use))
```

[ ]:
```
# create a pipeline with a single stage - the assembler. Fit the pipeline to␣
↪your data and create the transformed dataframe and name it␣
↪'modified_data_sdf'.
```

[ ]:
```
# converting string datatype to Integer for pipeline
```

```
final_walmart_sdf = final_walmart_sdf.withColumn("Dept",␣
↪final_walmart_sdf['Dept'].cast('int'))
final_walmart_sdf = final_walmart_sdf.withColumn("IsHoliday",␣
↪final_walmart_sdf['IsHoliday'].cast('int'))
```

[ ]:
```
from pyspark.ml import Pipeline

pipeline = Pipeline(stages=[assembler])
model = pipeline.fit(final_walmart_sdf)
modified_data_sdf = model.transform(final_walmart_sdf)
modified_data_sdf
```

[ ]:
```
import numpy as np
import pandas as pd
import json
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import cm
from datetime import datetime
import glob
import seaborn as sns
import re
import os
import datetime as dt
```

[ ]:
```
#Print results

pipeline_stat = pd.DataFrame(modified_data_sdf.take(5),␣
↪columns=modified_data_sdf.columns)
print('check_pipeline', (pipeline_stat.columns.values,␣
↪pipeline_stat['features'][0].size))
```

[ ]:
```
#create our train and test sets. Let us, split into an 80-20 ratio between the␣
↪train and test sets. Name these 'train_sdf' and 'test_sdf'
```

[ ]:
```
train_sdf,test_sdf = modified_data_sdf.randomSplit([0.8, 0.2])
```

[ ]:
```
# Print results
print('check_split', (train_sdf.count(), test_sdf.count()))
```

[ ]:
```
# Renamed Column Store as Label

train_sdf = train_sdf.withColumnRenamed("Store", "label")
train_sdf
test_sdf = train_sdf.withColumnRenamed("Store", "label")
test_sdf
```

[ ]:
```
# Import the Library
from pyspark.ml.classification import LogisticRegression
```

```
# Logistic Regression Features Training the model #

lr = LogisticRegression(maxIter=3, regParam=0.2, elasticNetParam=0)

# Train model with Training Data
lrModel = lr.fit(train_sdf)
```

[ ]:
```
# Model Prediction
predictions = lrModel.transform(test_sdf)
predictions
```

[ ]:
```
### Accuracy of Logistic Regression ###

# Import Libraries
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(labelCol="label",predictionCol="prediction")
evaluator.evaluate(predictions)
```

[ ]:
```
# train Linear Regression model to our data and predict. This prediction should be based on Spark ML's linear regression.
#Create a model using this library, fit the training data. Afterwards, print the summary stats of the model, i.e, the RMSE error, R2 score
#In this section, we will train the model without any regularization!
from pyspark.ml.regression import LinearRegression

# Add your code here

lr = LinearRegression()
lr_model = lr.fit(train_sdf)
lr_model.transform(test_sdf)
```

[ ]:
```
trainingSum = lr_model.summary

print("RMSE: %f" % trainingSum.rootMeanSquaredError)
print("r2: %f" % trainingSum.r2)
```

[ ]:
```
#Let us investigate that if the model actually overfits the training data.

#Predict the views for your test data (Note: it is called 'transform' in spark ml). Evaluate the performance using 'RegressionEvaluator' in the Spark ML Regression library. Name prediction column as 'prediction'.

predictions = lr_model.transform(test_sdf);
predictions
```

[ ]:
```
from pyspark.ml.evaluation import RegressionEvaluator
```

```
# Task: Compute RMSr on the test set

evaluator = RegressionEvaluator(
labelCol = "label", predictionCol="prediction", metricName="rmse")
test_rmse_orig = evaluator.evaluate(predictions)
```

[ ]:
```
#Print results here

predictions_to_print = predictions.toPandas()
lranswer = [test_rmse_orig, predictions_to_print['prediction'][0:50],␣
 ↪predictions_to_print['label'][0:50]]
print('result_lr_test', lranswer)
```

[ ]:
```
#implement regularization to avoid overfitting. you can try different␣
 ↪regularization parameters, e.g., try LASSO (L1), Ridge (L2) and elastic net␣
 ↪(combination of L1 and L2).

#Try different regularization hyperparameters to initialize three different␣
 ↪regularized linear regression models. Compare these regularization methods␣
 ↪with each other and the non-regularized method above.

# Compute predictions using each of the models

l1_predictions = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=1)
l2_predictions = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0)
elastic_net_predictions = LinearRegression(maxIter=10, regParam=0.
 ↪3,elasticNetParam=0.8)

l1_predictionsf = l1_predictions.fit(train_sdf)
l2_predictionsf = l2_predictions.fit(train_sdf)
elastic_net_predictionsf = elastic_net_predictions.fit(train_sdf)

l1_predictionst = l1_predictionsf.evaluate(test_sdf)
l1_predictionst = l2_predictionsf.evaluate(test_sdf)
elastic_net_predictionst = elastic_net_predictionsf.evaluate(test_sdf)

# Calculate the root mean squared error (RMSE) on test set for each of your␣
 ↪models

test_rmse_l1 = l1_predictionst.rootMeanSquaredError
test_rmse_l2 = l1_predictionst.rootMeanSquaredError
test_rmse_elastic = elastic_net_predictionst.rootMeanSquaredError
```

[ ]:
```
# Print your results here

result = [test_rmse_l1, test_rmse_l2, test_rmse_elastic]
print('result_lr_all', result)
```

```
[ ]: #### Accuracy calculation with Random Forest Model ####
from pyspark.ml.classification import RandomForestClassifier

## Create an initial RandomForest model.

rf = RandomForestClassifier(labelCol = "label", \
                            featuresCol = "features", \
                            numTrees = 100, \
                            maxDepth = 4, \
                            maxBins = 32)

# Train model with Training Data

rfModel = rf.fit(train_sdf)
```

```
[ ]: ### Model Prediction ###

predictions = rfModel.transform(test_sdf)

predictions.filter(predictions['prediction'] == 0) \
    .
 ↪select("Dept","Weekly_Sales","IsHoliday","label","prediction","probability")␣
 ↪\
    .orderBy("probability", ascending = False) \
    .show(n = 10, truncate = 30)
```

```
[ ]: ##### Accuracy Claculation with Random Forest #####

evaluator = MulticlassClassificationEvaluator(predictionCol = "prediction")
evaluator.evaluate(predictions)
```

```
[ ]: !wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('Big_Data_Technical_Project_Using_Spark_DataBricks_Kowsik_B.ipynb')
```

```
--2021-03-07 14:26:25--  https://raw.githubusercontent.com/brpy/colab-
pdf/master/colab_pdf.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.109.133, 185.199.111.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1864 (1.8K) [text/plain]
Saving to: colab_pdf.py

colab_pdf.py        100%[===================>]   1.82K  --.-KB/s    in 0s

2021-03-07 14:26:26 (38.1 MB/s) - colab_pdf.py saved [1864/1864]
```

```
Mounted at /content/drive/

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
```

BDA_Technical_Project_Data_Visualization_Kowsik_B

March 7, 2021

1 This is to visualize the Wallmart Sells Data

This lab aims to utilize with Pandas libraries for Big Data Analytics Technical Project.

The lab primarily focuses to i) data analysis ii) preparing datasets to analyze, plot, and feed to machine learning classifiers.

This lab requires to analyze three datasets under the given sections.

Used DataSets are: 1. features.csv 2. stores.csv 3. train.csv

[ ]:
```
# Importing standrad libaries

import numpy as np
import pandas as pd
import json
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import cm
from datetime import datetime
import glob
import seaborn as sns
import re
import os
import datetime as dt
```

[ ]:
```
%%capture
!apt install libkrb5-dev
!wget https://www-us.apache.org/dist/spark/spark-3.0.2/spark-3.0.2-bin-hadoop3.
↪2.tgz
!tar xf spark-3.0.2-bin-hadoop3.2.tgz
!pip install findspark
!pip install sparkmagic
!pip install pyspark
!pip install pyspark --user
```

[ ]:
```
!apt install libkrb5-dev
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

```
libkrb5-dev is already the newest version (1.16-2ubuntu0.2).
0 upgraded, 0 newly installed, 0 to remove and 29 not upgraded.
```

[ ]:
```
import os
os.environ['SPARK_HOME'] = "/content/spark-3.0.2-bin-hadoop3.2"
import pyspark
from pyspark.sql import SQLContext
from pyspark.sql import SparkSession
from pyspark.sql.types import *
import pyspark.sql.functions as F
from pyspark import SparkFiles
from pyspark.sql.types import StringType, IntegerType, DoubleType, StructField,␣
↪StructType, ArrayType, MapType
```

[ ]:
```
# Reading the Features dataset - features_df
features_df = pd.read_csv('/content/features.csv')
# Reading the Train dataset - train_df
train_df = pd.read_csv('/content/train.csv')
# Reading the Stores dataset - stores_df
stores_df = pd.read_csv('/content/stores.csv')
```

[ ]:
```
# Checking for null values in the Data sets
train_df.isnull().sum()
features_df.isnull().sum()
stores_df.isnull().sum()
```

[ ]:
```
Store    0
Type     0
Size     0
dtype: int64
```

[ ]:
```
# dataset columns values for Train Data Frame
pd.DataFrame(train_df.dtypes, columns=['Type']).T
```

[ ]:
```
      Store   Dept    Date Weekly_Sales IsHoliday
Type  int64  int64  object      float64      bool
```

[ ]:
```
# For Features Data Frame
pd.DataFrame(features_df.dtypes, columns=['Type']).T
```

[ ]:
```
      Store    Date Temperature  ...      CPI Unemployment IsHoliday
Type  int64  object     float64  ...  float64      float64      bool

[1 rows x 12 columns]
```

[ ]:
```
# For Stores Data Frame
pd.DataFrame(stores_df.dtypes, columns=['Type']).T
```

[ ]:
```
      Store    Type   Size
Type  int64  object  int64
```

2 Data Cleaning Activity on Spark Data Frame

Drop duplicate rows
Drop Null rows
Instead Dropping rows that have values as 'NaN' or 'NA' will replace any empty cells with 0 instead.
Drop columns those aren't neccessary for the rest of the lab.
Rename Columns as necessary
Save the result to final_walmart_df data frame

[ ]:
```
# Drop duplicates from all 3 data frames

train_df = train_df.dropna()
features_df = features_df.dropna()
stores_df = stores_df.dropna()
```

[ ]:
```
%%time
features_df.describe()
```

```
CPU times: user 23 ms, sys: 1.19 ms, total: 24.2 ms
Wall time: 29.5 ms
```

[ ]:

	Store	Temperature	...	CPI	Unemployment
count	2069.000000	2069.000000	...	2069.000000	2069.000000
mean	20.386660	52.516979	...	175.587370	7.252611
std	12.076174	18.483053	...	39.963914	1.684774
min	1.000000	-7.290000	...	129.816710	3.684000
25%	10.000000	38.100000	...	137.423897	6.162000
50%	20.000000	51.420000	...	189.707605	7.191000
75%	29.000000	66.610000	...	219.970560	8.256000
max	45.000000	95.910000	...	228.976456	12.890000

[8 rows x 10 columns]

[ ]:
```
%%time
train_df.describe()
```

```
CPU times: user 45 ms, sys: 4.66 ms, total: 49.7 ms
Wall time: 53.7 ms
```

[ ]:

	Store	Dept	Weekly_Sales
count	421570.000000	421570.000000	421570.000000
mean	22.200546	44.260317	15981.258123
std	12.785297	30.492054	22711.183519
min	1.000000	1.000000	-4988.940000
25%	11.000000	18.000000	2079.650000
50%	22.000000	37.000000	7612.030000
75%	33.000000	74.000000	20205.852500
max	45.000000	99.000000	693099.360000

[ ]:
```
%%time
stores_df.describe()
```

```
CPU times: user 9.14 ms, sys: 0 ns, total: 9.14 ms
Wall time: 13.9 ms
```

[ ]:

	Store	Size
count	45.000000	45.000000
mean	23.000000	130287.600000
std	13.133926	63825.271991
min	1.000000	34875.000000
25%	12.000000	70713.000000
50%	23.000000	126512.000000
75%	34.000000	202307.000000
max	45.000000	219622.000000

[ ]:
```
# doing this task using SQL approach using same 3 Pandas Data Frames

# Importing libraries
import sqlite3
conn = sqlite3.connect('local.db')
# Query to get data count from features_df
features_df.to_sql ("features", conn, if_exists="replace", index=False)
features_df_count = pd.read_sql_query("select count(*) as 'Features_DF_Cnt'␣
↪from features", conn)
features_df_count

# Query to get data count from stores_df
stores_df.to_sql ("stores", conn, if_exists="replace", index=False)
stores_df_count = pd.read_sql_query("select count(*) as 'Stores_DF_Cnt' from␣
↪stores", conn)
stores_df_count

# Query to get data count from train_df
train_df.to_sql ("train", conn, if_exists="replace", index=False)
train_df_count = pd.read_sql_query("select count(*) as 'Train_DF_Cnt' from␣
↪train", conn)
train_df_count

count_df = pd.concat([train_df_count,stores_df_count,features_df_count], axis =␣
↪1)
count_df
```

[ ]:

	Train_DF_Cnt	Stores_DF_Cnt	Features_DF_Cnt
0	421570	45	2069

[ ]:
```
# Drop rows with null value from all 3 data sets and count all datasets
# Using pandas
```

```
train_df_drop_na = train_df.dropna()
features_df_drop_na = features_df.dropna()
stores_df_drop_na = stores_df.dropna()

# Perform a count again after dropping NA values from all 3 datasets.
# Query to get data count from features_df
features_df_drop_na.to_sql ("features", conn, if_exists="replace", index=False)
features_df_count = pd.read_sql_query("select count(*) as 'Features_DF_Cnt'␣
↪from features", conn)

# Query to get data count from stores_df
stores_df_drop_na.to_sql ("stores", conn, if_exists="replace", index=False)
stores_df_count = pd.read_sql_query("select count(*) as 'Stores_DF_Cnt' from␣
↪stores", conn)

# Query to get data count from train_df
train_df_drop_na.to_sql ("train", conn, if_exists="replace", index=False)
train_df_count = pd.read_sql_query("select count(*) as 'Train_DF_Cnt' from␣
↪train", conn)

count_df = pd.concat([train_df_count,stores_df_count,features_df_count], axis =␣
↪1)
count_df
```

[ ]:

```
   Train_DF_Cnt  Stores_DF_Cnt  Features_DF_Cnt
0        421570             45             2069
```

[ ]:

```
train_df_drop_na.head(2)
features_df_drop_na.head(2)
#stores_df_drop_na = stores_df.dropna()
```

[ ]:

```
    Store        Date  Temperature  ...         CPI  Unemployment  IsHoliday
92      1  2011-11-11        59.11  ...  217.998085         7.866      False
93      1  2011-11-18        62.25  ...  218.220509         7.866      False

[2 rows x 12 columns]
```

[ ]:

```
# Merge all 3 data frames to a final data frame called - final_walmart_df
# Perform an inner join between the train_df, stores_df and features_df on the␣
↪Store, Date, IsHoliday column.

# merge on train and features data frame
merge_tr_fr_df = train_df_drop_na.merge(features_df_drop_na, how="left",␣
↪on=["Store","Date","IsHoliday"])
merge_tr_fr_df.head(3)

# merge on train and stores data frame
```

```
final_walmart_df = merge_tr_fr_df.merge(stores_df_drop_na, how="left",␣
 ↪on=["Store"])
display(final_walmart_df)

# Check how our final final_walmart_df looks like
print("Rows & Columns: ", final_walmart_df.shape, "\nAll columns in the␣
 ↪final_walmart_df Data Frame: ", final_walmart_df.columns.tolist())

# Replace the NaN values in markdown with 0
final_walmart_df = final_walmart_df.fillna(0.0)
final_walmart_df
```

```
        Store  Dept        Date  Weekly_Sales  ...  CPI  Unemployment  Type    Size
0           1     1  05-02-2010      24924.50  ...  NaN           NaN     A  151315
1           1     1  12-02-2010      46039.49  ...  NaN           NaN     A  151315
2           1     1  19-02-2010      41595.55  ...  NaN           NaN     A  151315
3           1     1  26-02-2010      19403.54  ...  NaN           NaN     A  151315
4           1     1  05-03-2010      21827.90  ...  NaN           NaN     A  151315
...       ...   ...         ...           ...  ...  ...           ...   ...     ...
421565     45    98  28-09-2012        508.37  ...  NaN           NaN     B  118221
421566     45    98  05-10-2012        628.10  ...  NaN           NaN     B  118221
421567     45    98  12-10-2012       1061.02  ...  NaN           NaN     B  118221
421568     45    98  19-10-2012        760.01  ...  NaN           NaN     B  118221
421569     45    98  26-10-2012       1076.80  ...  NaN           NaN     B  118221

[421570 rows x 16 columns]


Rows & Columns:  (421570, 16)
All columns in the final_walmart_df Data Frame:  ['Store', 'Dept', 'Date',
'Weekly_Sales', 'IsHoliday', 'Temperature', 'Fuel_Price', 'MarkDown1',
'MarkDown2', 'MarkDown3', 'MarkDown4', 'MarkDown5', 'CPI', 'Unemployment',
'Type', 'Size']
```

[ ]:
```
        Store  Dept        Date  Weekly_Sales  ...  CPI  Unemployment  Type
Size
0           1     1  05-02-2010      24924.50  ...  0.0           0.0     A
151315
1           1     1  12-02-2010      46039.49  ...  0.0           0.0     A
151315
2           1     1  19-02-2010      41595.55  ...  0.0           0.0     A
151315
3           1     1  26-02-2010      19403.54  ...  0.0           0.0     A
151315
4           1     1  05-03-2010      21827.90  ...  0.0           0.0     A
151315
...       ...   ...         ...           ...  ...  ...           ...   ...
```

```
...
421565     45    98  28-09-2012        508.37  ...  0.0           0.0     B
118221
421566     45    98  05-10-2012        628.10  ...  0.0           0.0     B
118221
421567     45    98  12-10-2012       1061.02  ...  0.0           0.0     B
118221
421568     45    98  19-10-2012        760.01  ...  0.0           0.0     B
118221
421569     45    98  26-10-2012       1076.80  ...  0.0           0.0     B
118221

[421570 rows x 16 columns]
```

[ ]:
```
# Standard plotly imports
!pip install cufflinks plotly
!pip install chart_studio
```

```
Requirement already satisfied: cufflinks in /usr/local/lib/python3.7/dist-
packages (0.17.3)
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages
(4.4.1)
Requirement already satisfied: setuptools>=34.4.1 in /usr/local/lib/python3.7
/dist-packages (from cufflinks) (54.0.0)
Requirement already satisfied: ipywidgets>=7.0.0 in /usr/local/lib/python3.7
/dist-packages (from cufflinks) (7.6.3)
Requirement already satisfied: ipython>=5.3.0 in /usr/local/lib/python3.7/dist-
packages (from cufflinks) (5.5.0)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.7/dist-
packages (from cufflinks) (1.15.0)
Requirement already satisfied: colorlover>=0.2.1 in /usr/local/lib/python3.7
/dist-packages (from cufflinks) (0.3.0)
Requirement already satisfied: pandas>=0.19.2 in /usr/local/lib/python3.7/dist-
packages (from cufflinks) (1.1.5)
Requirement already satisfied: numpy>=1.9.2 in /usr/local/lib/python3.7/dist-
packages (from cufflinks) (1.19.5)
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.7/dist-
packages (from plotly) (1.3.3)
Requirement already satisfied: ipykernel>=4.5.1 in /usr/local/lib/python3.7
/dist-packages (from ipywidgets>=7.0.0->cufflinks) (4.10.1)
Requirement already satisfied: jupyterlab-widgets>=1.0.0; python_version >=
"3.6" in /usr/local/lib/python3.7/dist-packages (from
ipywidgets>=7.0.0->cufflinks) (1.0.0)
Requirement already satisfied: widgetsnbextension~=3.5.0 in
/usr/local/lib/python3.7/dist-packages (from ipywidgets>=7.0.0->cufflinks)
(3.5.1)
Requirement already satisfied: traitlets>=4.3.1 in /usr/local/lib/python3.7
/dist-packages (from ipywidgets>=7.0.0->cufflinks) (5.0.5)
```

```
Requirement already satisfied: nbformat>=4.2.0 in /usr/local/lib/python3.7/dist-
packages (from ipywidgets>=7.0.0->cufflinks) (5.1.2)
Requirement already satisfied: simplegeneric>0.8 in /usr/local/lib/python3.7
/dist-packages (from ipython>=5.3.0->cufflinks) (0.8.1)
Requirement already satisfied: decorator in /usr/local/lib/python3.7/dist-
packages (from ipython>=5.3.0->cufflinks) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.7/dist-
packages (from ipython>=5.3.0->cufflinks) (0.7.5)
Requirement already satisfied: pexpect; sys_platform != "win32" in
/usr/local/lib/python3.7/dist-packages (from ipython>=5.3.0->cufflinks) (4.8.0)
Requirement already satisfied: pygments in /usr/local/lib/python3.7/dist-
packages (from ipython>=5.3.0->cufflinks) (2.6.1)
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.4 in
/usr/local/lib/python3.7/dist-packages (from ipython>=5.3.0->cufflinks) (1.0.18)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-
packages (from pandas>=0.19.2->cufflinks) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas>=0.19.2->cufflinks) (2.8.1)
Requirement already satisfied: tornado>=4.0 in /usr/local/lib/python3.7/dist-
packages (from ipykernel>=4.5.1->ipywidgets>=7.0.0->cufflinks) (5.1.1)
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.7/dist-
packages (from ipykernel>=4.5.1->ipywidgets>=7.0.0->cufflinks) (5.3.5)
Requirement already satisfied: notebook>=4.4.1 in /usr/local/lib/python3.7/dist-
packages (from widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (5.3.1)
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.7
/dist-packages (from traitlets>=4.3.1->ipywidgets>=7.0.0->cufflinks) (0.2.0)
Requirement already satisfied: jupyter-core in /usr/local/lib/python3.7/dist-
packages (from nbformat>=4.2.0->ipywidgets>=7.0.0->cufflinks) (4.7.1)
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in
/usr/local/lib/python3.7/dist-packages (from
nbformat>=4.2.0->ipywidgets>=7.0.0->cufflinks) (2.6.0)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.7/dist-
packages (from pexpect; sys_platform != "win32"->ipython>=5.3.0->cufflinks)
(0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/dist-packages
(from prompt-toolkit<2.0.0,>=1.0.4->ipython>=5.3.0->cufflinks) (0.2.5)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.7/dist-
packages (from jupyter-client->ipykernel>=4.5.1->ipywidgets>=7.0.0->cufflinks)
(22.0.3)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.7/dist-
packages (from
notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks)
(5.6.1)
Requirement already satisfied: Send2Trash in /usr/local/lib/python3.7/dist-
packages (from
notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks)
(1.5.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages
```

```
(from notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks)
(2.11.3)
Requirement already satisfied: terminado>=0.8.1 in /usr/local/lib/python3.7
/dist-packages (from
notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks)
(0.9.2)
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.7
/dist-packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipyw
idgets>=7.0.0->cufflinks) (0.3)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.7/dist-
packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets
>=7.0.0->cufflinks) (0.6.0)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.7
/dist-packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipyw
idgets>=7.0.0->cufflinks) (0.8.4)
Requirement already satisfied: bleach in /usr/local/lib/python3.7/dist-packages
(from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->
cufflinks) (3.3.0)
Requirement already satisfied: testpath in /usr/local/lib/python3.7/dist-
packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets
>=7.0.0->cufflinks) (0.4.4)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.7
/dist-packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipyw
idgets>=7.0.0->cufflinks) (1.4.3)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7
/dist-packages (from jinja2->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidg
ets>=7.0.0->cufflinks) (1.1.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-
packages (from bleach->nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ip
ywidgets>=7.0.0->cufflinks) (20.9)
Requirement already satisfied: webencodings in /usr/local/lib/python3.7/dist-
packages (from bleach->nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ip
ywidgets>=7.0.0->cufflinks) (0.5.1)
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.7
/dist-packages (from packaging->bleach->nbconvert->notebook>=4.4.1->widgetsnbext
ension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (2.4.7)
Collecting chart_studio
  Downloading https://files.pythonhosted.org/packages/ca/ce/330794a6b6ca4b
9182c38fc69dd2a9cbff60fd49421cb8648ee5fee352dc/chart_studio-1.1.0-py3-none-
any.whl (64kB)
     || 71kB 3.2MB/s
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.7
/dist-packages (from chart_studio) (1.3.3)
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages
(from chart_studio) (4.4.1)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-
packages (from chart_studio) (2.23.0)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages
```

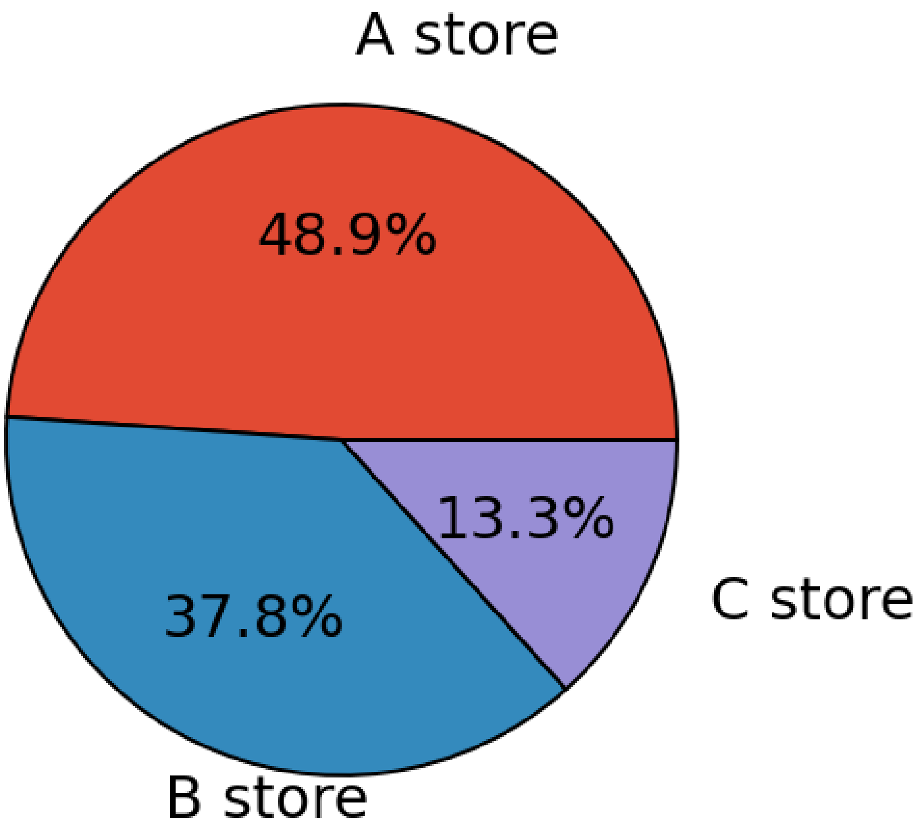
```
(from chart_studio) (1.15.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7
/dist-packages (from requests->chart_studio) (2020.12.5)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7
/dist-packages (from requests->chart_studio) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from requests->chart_studio) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests->chart_studio) (2.10)
Installing collected packages: chart-studio
Successfully installed chart-studio-1.1.0
```

[ ]:
```
import plotly
import chart_studio.plotly as py
import plotly.figure_factory as ff
import plotly.graph_objs as go
from plotly.offline import iplot, init_notebook_mode
# Using plotly + cufflinks in offline mode
import cufflinks
cufflinks.go_offline(connected=True)
init_notebook_mode(connected=True)
```

3 Data Visualisation Of Wallmart data

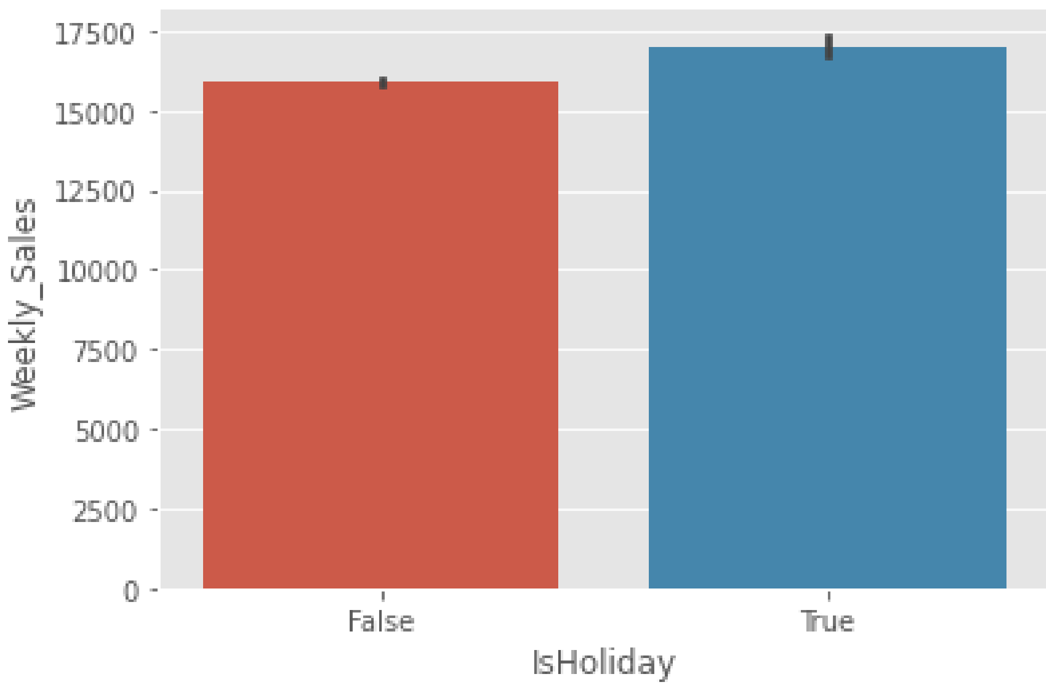
[ ]:
```
# Store Type Vs. Weekly Sales

sorted_type = stores_df.groupby('Type')
plt.style.use('ggplot')
labels=['A store','B store','C store']
sizes=sorted_type.describe()['Size'].round(1)
sizes=[(22/(17+6+22))*100,(17/(17+6+22))*100,(6/(17+6+22))*100] # convert to␣
 ↪the proportion
fig, axes = plt.subplots(1,1, figsize=(10,10))
wprops={'edgecolor':'black',
      'linewidth':2}
tprops = {'fontsize':30}
axes.pie(sizes,
        labels=labels,
        explode=(0.0,0,0),
        autopct='%1.1f%%',
        pctdistance=0.6,
        labeldistance=1.2,
        wedgeprops=wprops,
        textprops=tprops,
        radius=0.8,
        center=(0.5,0.5))
```

```
plt.show()
```

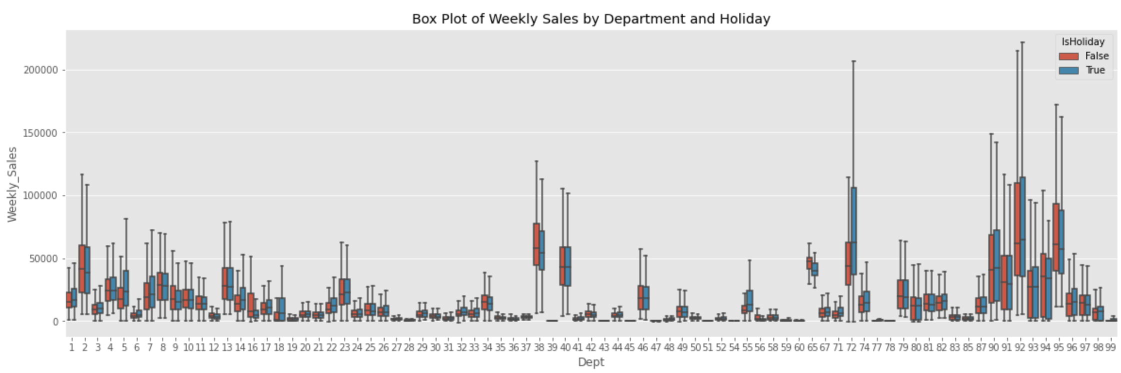


[ ]:
```
# Weekly Sales on holidays and non-holidays

holiday = final_walmart_df['Weekly_Sales'].loc[final_walmart_df['IsHoliday']==␣
↪True]       # Weekly Sales in Holidays
non_holiday = final_walmart_df['Weekly_Sales'].
↪loc[final_walmart_df['IsHoliday']== False] # Weekly Sales in Non-holidays.
sns.barplot(x='IsHoliday', y='Weekly_Sales', data=final_walmart_df)
```

[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5fed794a10>
```




```
[ ]: # Weekly Sales by Department and Holiday

data_11= pd.concat([final_walmart_df['Dept'], final_walmart_df['Weekly_Sales'],␣
 ↪final_walmart_df['IsHoliday']], axis=1)
plt.figure(figsize=(20,6))
plt.title('Box Plot of Weekly Sales by Department and Holiday')
fig = sns.boxplot(x='Dept', y='Weekly_Sales', data=data_11, showfliers=False,␣
 ↪hue="IsHoliday")
```



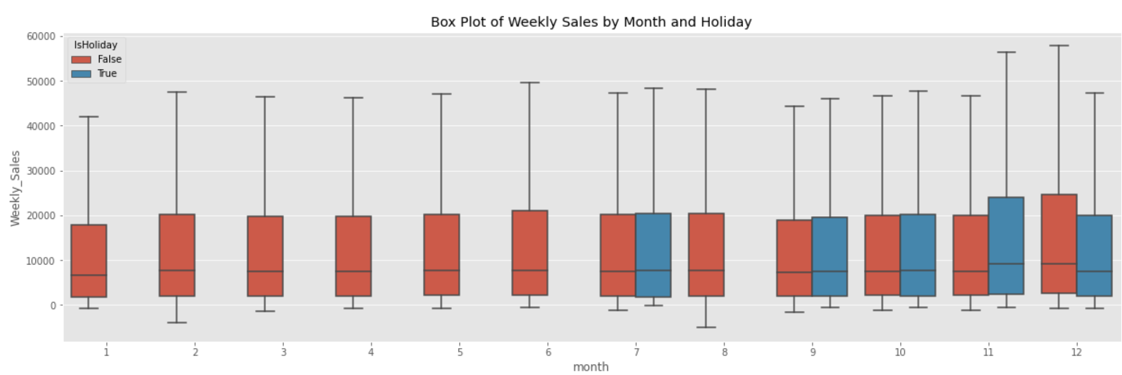

```
[ ]: final_walmart_df.dtypes
```

[ ]:
```
Store              int64
Dept               int64
Date              object
Weekly_Sales     float64
IsHoliday           bool
Temperature      float64
Fuel_Price       float64
MarkDown1        float64
MarkDown2        float64
MarkDown3        float64
MarkDown4        float64
MarkDown5        float64
CPI              float64
Unemployment     float64
Type              object
Size               int64
dtype: object
```

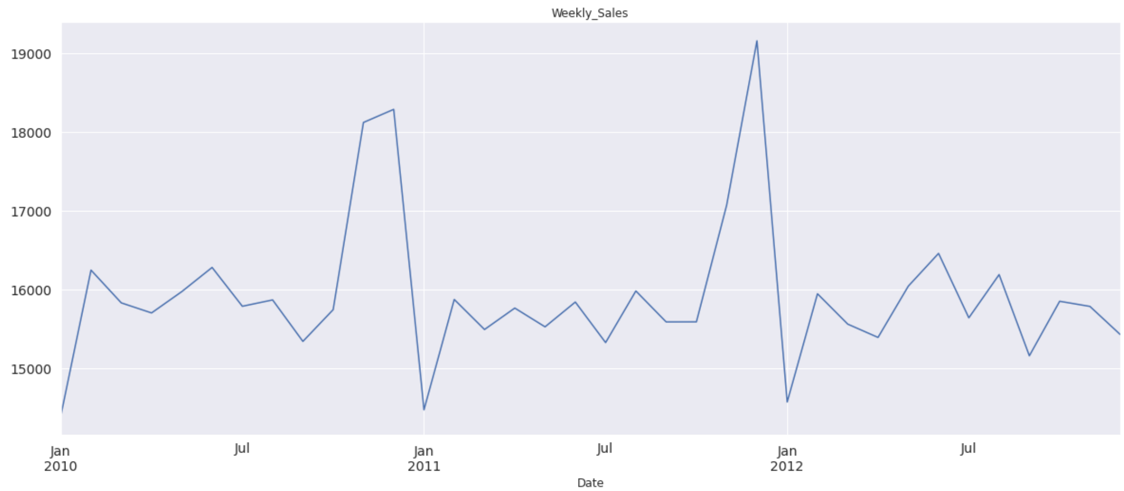
[ ]:
```
final_walmart_df['Date'] = pd.to_datetime(final_walmart_df['Date'])
final_walmart_df['month'] = pd.DatetimeIndex(final_walmart_df['Date']).month
final_walmart_df.head(2)
```

[ ]:
```
   Store  Dept       Date  Weekly_Sales  ...  Unemployment  Type    Size  month
0      1     1 2010-05-02      24924.50  ...           0.0     A  151315      5
1      1     1 2010-12-02      46039.49  ...           0.0     A  151315     12

[2 rows x 17 columns]
```

[ ]:
```
# Month wise Weekly Sales visualization
df_m = pd.concat([final_walmart_df['month'], final_walmart_df['Weekly_Sales'],␣
 ↪final_walmart_df['IsHoliday']], axis=1)
plt.figure(figsize = (20,6))

plt.title('Box Plot of Weekly Sales by Month and Holiday')
fig = sns.boxplot(x = 'month', y = 'Weekly_Sales', data = df_m, showfliers =␣
 ↪False, hue = 'IsHoliday')
```



[ ]:
```
# Below plot is general idea of how the Weekly Sales has been so far

final_walmart_df.index = final_walmart_df.Date
final_walmart_df = final_walmart_df.drop('Date', axis=1)
final_walmart_df = final_walmart_df.resample('MS').mean()

train_dt = final_walmart_df['Weekly_Sales']
# Plot of Weekly_Sales with respect to years in train

train_dt.plot(figsize = (20,8), title = 'Weekly_Sales', fontsize = 14)
plt.show()
```



[ ]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('BDA_Technical_Project_Data_Visualization_Kowsik_B.ipynb')
```

```
--2021-03-07 17:42:42--  https://raw.githubusercontent.com/brpy/colab-
pdf/master/colab_pdf.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.109.133, 185.199.111.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1864 (1.8K) [text/plain]
Saving to: colab_pdf.py

colab_pdf.py        100%[===================>]   1.82K  --.-KB/s    in 0s

2021-03-07 17:42:42 (46.8 MB/s) - colab_pdf.py saved [1864/1864]
```

```
Mounted at /content/drive/

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
```