Table of Contents

[Introduction 3](#_Toc513147897)

[1.1 Executive Summary 3](#_Toc513147898)

[1.2 Resources 3](#_Toc513147899)

[2. ETL PROCESS 4](#_Toc513147900)

[2.1 Understanding the Data 4](#_Toc513147901)

[2.2 Data Cleansing 4](#_Toc513147902)

[2.3 Data Transformation 5](#_Toc513147903)

[2.4 Data Loading 5](#_Toc513147904)

[APPENDICES 6](#_Toc513147905)

[APPENDIX – DATA DICTIONARY 6](#_Toc513147906)

[APPENDIX B – CODES 10](#_Toc513147907)

[APPENDIX C – DIAGRAMS 23](#_Toc513147908)

# Introduction

The purpose of the Extraction, Transformation, and Load (ETL) Specification Document is to capture details that pertain specifically to ETL development to be used by the developer as an aid in ETL development.

## Executive Summary

This document is prepared to explain the ETL process of United Super Markets data that consist of 5 flat files and one of which is scraped data. The aim was to examine the data, conduct the cleansing process and transform the data to make them ready to load to the NoSQL database, namely to Mongo DB. Item list file is updated with the new “UPC” detail by comparing the existing item list with the extracted item list details from FTP server. Details for extracting, transforming and loading processes will be explained in this document. Finally, data dictionary, codes, and diagrams about each process will be given to explain the whole process.

## Resources

The resources of this ETL process are 5 flat files; namely Item List, Sales Details, Customer List, Item Attributes, and Store Location. These files were provided by United Super Markets via a cloud link. To process the data, Python (a programming language) and Mongo DB (Cross-platform document-oriented NoSQL database) are used as software.

**Data File Information**

|  |  |
| --- | --- |
| Table/Collection | File Name |
| Item List | item\_list.txt |
| Sales Details | sls\_dtl.txt |
| Customer List | customer\_List.txt |
| Item Attributes | Item\_Attr.txt |
| Store Location | Store\_location.txt (Scrape data) |

# ETL PROCESS

[This section outlines a more detailed description of the processes that are currently utilized.]

## Understanding the Data

Before starting data process, it is important to understand the data structure and set the objectives that needed to be accomplished. Each file is received as a text file that is “|” delimited with no fixed length. After receiving files, we tried to understand the data by reading the data dictionary and examining the tables/collections. We observed that files need to be cleansed in the first place. The explanation of each table/collection (column names and descriptions) is given in Appendix A (Data Dictionary).

## Data Cleansing

Python programming language program was used for cleansing process. Each txt files have different issues that need to be handle. As a general approach, columns with 1) junk characters, 2) null values are excluded while reading the collection. In addition to this, some of the files have unnecessary data at first few rows; these are also excluded. Another issue was about encoding. As the collections have some certain characters that the generic utf-8 encoding approach did not work properly; to make collections readable properly, in terms of encoding approach, we used latin\_1 which is a universal format for windows. The Python codes are provided in Appendix B.

Beyond the general cleansing strategies, specific data cleansing process of each file (Table/Collection) is explained below:

|  |  |  |
| --- | --- | --- |
| Collection Name | File Name | Cleansing Process |
| Item List | item\_list.txt | Null values columns excluded (after last “|” delimiter) |
| Sales Details | sls\_dtl.txt | First 3 characters of the first column was excluded (Junk Characters)  Null values columns were excluded (after last “|” delimiter) |
| Customer List | customer\_List.txt | Null values columns were excluded (after last “|” delimiter)  Null values rows (if applicable for all columns) were excluded  Quoting characters were ignored during the process |
| Item Attributes | Item\_Attr.txt | First 3 rows (unnecessary information) were excluded  Null values columns were excluded (after last “|” delimiter) |
| Item Hierarchy | (Source is item list) | Extracted hierarchy details for each “UPC” and item from Item List file. |
| Store Location | Store\_location.txt | NA |

## Data Transformation

After data cleansing, transformation process is applied to some of the data. As the default date format of Mango DB is different from the original .txt data files, we transformed the date format from central time to GMT by using Python to generalize the date format. In addition to this, store location data was provided as key value. For further processing, to make it easy to follow, we transformed store location file from key value to table structure.

Details about transformation is given below:

|  |  |  |
| --- | --- | --- |
| Collection Name | File Name | Transformation Process |
| Item List | item\_list.txt | NA |
| Sales Details | sls\_dtl.txt | Parse the Dates. Dates are loaded as GMT in Mongo DB |
| Customer List | customer\_List.txt | NA |
| Item Attributes | Item\_Attr.txt | Parse the Dates. Dates are loaded as GMT in Mongo DB |
| Item Hierarchy | (Source is item list) | By using hierarchy given in Appendix A, data was transformed to be loaded to Mongo DB |
| Store Location | store\_location | Changing the data type from key value to table structure |

## Data Loading

Once the cleansing and transformation processes are done, tables/collections were loaded to a NoSQL database called Mango DB. To do so, first we have received the schema design of *“Mango DB from US NoSQL”* Team. Then we applied this schema while loading the tables/collections by using Python. To update the new “UPC”s, existing item list file is compared with the new item list file extracted from FTP server and updated new item list file is loaded to Mongo DB.

The Diagrams of the two main task / process are represented in Appendix C.

## APPENDICES

## APPENDIX – DATA DICTIONARY

**Item List file**

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Data type** |
| UPC | ID# for individual products. Two standardized systems are included (the 12 digit EAN system and the 10 digit UPC-A system) and a number of shorter forms of UPC codes that are generated within the stores by various in-house activity centers such as bakeries, delis, etc. The UPC variable is the key for merging the transaction files and item files (reported in both types of files). | Nominal |
| Status | Individual products may be: **0 - Invalid Item, 1 - Active, 2 – Suspended or 3 - Deleted.** | Nominal |
| RootID | Today this is a randomly generated number. However, it can be used to roll up multiple items into a single item root. For example, if you have a 12oz bottle of coke, and they do a "bonus buy" of 50% more free and temporarily put an 18oz bottle in its place, you could tie both product UPC's to the same RootID. We do not do this today...every UPC will have its own item root. In the future this could change. | Nominal |
| LongDes | A more complete description of a particular product/UPC item (e.g., KELLOGGS ALL BRAN CEREAL, \*Z\*TROPICANA PP ORG JCE L/ACID, etc.) | Nominal |
| ShortDes | A less complete description of a particular product /UPC item (KELLOGGS ALL BRAN CE, TROPICANA PP ORG JCE, etc.) | Nominal |
| ClassCode\* | Fourth broadest grouping of products/UPCs (Parent hierarchy of UPC) | Nominal |
| ClassDes\* | Verbal descriptions of \_\_ product classes (e.g., BOX COLD CEREALS, ORANGE. etc.) | Nominal |
| CategoryCode\* | Third broadest grouping of products/UPCs (parent hierarchy level of class) | Nominal |
| CategoryDes\* | Verbal Descriptions of product Categories (e.g., COLD CEREALS, etc.) | Nominal |
| FamilyCode\* | Second broadest grouping products/UPCs (Parent hierarchy level of category) | Nominal |
| FamilyDes\* | Verbal descriptions product Families\* (e.g., CEREALS, REFRIGERATED DRINKS/JUICES, etc.) | Nominal |
| DepartmentCode\* | Broadest grouping of products/UPCs (Parent hierarchy level of family) | Nominal |
| StoreBrand | It’s a **Y- Yes or N- No** attribute for each item | Nominal |
| ExtraDes | It’s the additional description of the item. This attribute is not uniform in nature. Sometimes, it contains county of origin, sometimes it consists of packaging information. | Nominal |

**\* List available below**

An Example of Product Grouping Hierarchy:

Level 1 DepartmentCode = 2

DepartmentDes = BAKERY

Level 2 FamilyCode = 4040

FamilyDes = COOKIES

Level 3 CategoryCode = 40400105

CategoryDes = GOURMET

Level 4 ClassCode = 40400105010

ClassDes = LABEL

Level 5 UPC = 00065477300659

**Sales Details file**

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Data type** |
| StoreNum | ID for stores. There are 50 + stores in the United system. StoreNum = 500 is a Pharmacy. The United system has 5 types of stores: Amigos, Albertsons, United, United Express, and Market Street | Nominal |
| Register | ID for check-out counters. **These numbers are unique only within stores.** | Nominal |
| TransNum | ID for specific customer transactions. These numbers range between 0 and 9999 and they **repeat** within stores within days. | Nominal |
| TransDate | Year-Month-Day for specific customer transaction (paid by customer). This variable can be converted to interval data. | Nominal |
| TransTime | Hour-Minute-Second for specific customer transaction (scanned, not paid). This variable can be converted to interval data. | Nominal |
| BusDate | Year-Month-Day for specific customer transaction (posted by store). This variable can be converted to interval data. | Nominal |
| UPC | ID# for individual products. Two standardized systems are included (the 12 digit EAN system and the 10 digit UPC-A system) and a number of shorter forms of UPC codes that are generated within the stores by various in-house activity centers such as bakeries, delis, etc. The UPC variable is the key for merging the transaction files and item files (reported in both types of files). | Nominal |
| DeptNum | ID assigned to each departments within each store | Nominal |
| ItemQuant | The number of **product units purchased**. For integer items (e.g., a box of cereal) the values will be 1.00, etc. However, for products sole by the pound or other measures the values may be a fraction (e.g., 1.53 pounds of bananas). | Ratio |
| WeightAmt | The amount of an item based on its weight. Example: Bananas | Ratio |
| SalesAmt |  | Ratio |
| CostAmt |  | Ratio |
| CashierNum | Unique ID code for a cashier (who may work at different registers) | Nominal |
| PriceType | Type of price promotion inside a store. | Nominal |
| ServiceType | Describes if the purchase was in-store or online | Nominal |
| TenderType | Payment type | Nominal |
| LoyaltyCardNumber | This is a unique customer number assigned to each of United’s customers participating in the Loyalty program | Nominal |

**Customer List file**

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Data type** |
| LoyaltyCardNum | This is a unique customer number assigned to each of United’s customers participating in the Loyalty program | Nominal |
| HouseholdNum | United assigned household numbers to their customers | Nominal |
| MemberFavStore | They also identified which store was the favorite store for its loyalty program customers | Nominal |
| City | The city the customers resided in | Nominal |
| State | The state the customers resided in | Nominal |
| ZipCode | The Zip codes of the customers | Nominal |

In addition to these files, we also get **Item Attribute file** sometimes.

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Data type** |
| UPC | ID# for individual products. Two standardized systems are included (the 12 digit EAN system and the 10 digit UPC-A system) and a number of shorter forms of UPC codes that are generated within the stores by various in-house activity centers such as bakeries, delis, etc. The UPC variable is the key for merging the transaction files and item files (reported in both types of files). | Nominal |
| ItemPosDes | A more complete description of a particular product/UPC item (e.g., KELLOGGS ALL BRAN CEREAL, \*Z\*TROPICANA PP ORG JCE L/ACID, etc.) | Nominal |
| ItemAttributeDes | Contains values like:  Made in Texas,  Nuval Score,  Gluten Free,  Organic Certified, etc. | Nominal |
| ItemAttributeValue | Contains both number scores and Yes/No values | Nominal |
| AttributeStartDate | Start Date of an attribute | Nominal |
| AttributeEndDate | End Date of an attribute | Nominal |

## APPENDIX B – CODES

**CODES for ETL**

import pandas as pd

import numpy as np

from pymongo import MongoClient

from datetime import datetime

from datetime import timedelta

import os

def cleanSalesTrx(path,filename):

"""

Sales Transaction file has few junk characters at the start of every row and empty column at the end.

This function removes the junk characters and empty column and creates a cleaner version of the file.

Return full file name to calling program.

"""

messyFile = open(path+filename,encoding='latin\_1')

tidyFile = open(path+'SalesTrxCln.txt','a')

for line in messyFile.readlines():

tidyFile.write(line[3:-2]+'\n')

tidyFile.close()

print('Sales Transaction File is cleaned')

return(tidyFile.name)

def extractSalesTrx(filename):

"""

This function extracts the sales transaction data from cleaned Sales file to python

It returns the sales transaction data frame

"""

sales\_cols = {'StoreNum':np.int64,

'Register':np.int64,

'TransNum':np.int64,

'TransDatetime(GMT)':'datetime64[ns]',

'TransDatetime(Local)':str,

'BusDate':'datetime64[ns]',

'UPC':str,

'ItemID':str,

'DeptNum':np.int64,

'ItemQuantity':np.float64,

'WeightAmt':np.float64,

'SalesAmt':np.float64,

'CostAmt':np.float64,

'CashierNum':np.int64,

'PriceType':str,

'ServiceType':str,

'TenderType':str,

'LoyaltyCardNumber':np.int64}

salesTrx = pd.read\_csv(filename,

sep='|',

header=None,

parse\_dates= [[3,4],5])

salesTrx=pd.concat([salesTrx.iloc[:,1:4],salesTrx.iloc[:,0],salesTrx.iloc[:,0].astype(str),salesTrx.iloc[:,4:]],axis=1)

salesTrx.columns=sales\_cols.keys()

salesTrx.fillna({'UPC':-999,

'LoyaltyCardNumber':-999},inplace=True)

salesTrx=salesTrx.astype(sales\_cols)

salesTrx['BusDate'] = salesTrx['BusDate']+timedelta(hours=12)

salesTrx.name = 'SalesTrx'

print(salesTrx.dtypes)

return(salesTrx)

def extractItemAttr(path,filename):

"""

This function extracts the sales transaction data from Item Attribute file to python

It returns the Item Atrribute data frame

"""

attr\_cols = {"UPC":str,

"ItemPosDes":str,

"ItemAttributeDes":str,

"ItemAttributeValue":str,

"AttributeStartDate":"datetime64[ns]",

"AttributeEndDate":"datetime64[ns]"

}

item\_att= pd.read\_csv(path+filename,

sep = '|',

skiprows=3,

header = None,

names=attr\_cols.keys(),

parse\_dates=[4,5])

item\_att = item\_att.astype(attr\_cols)

item\_att['AttributeStartDate'] = item\_att['AttributeStartDate']+timedelta(hours=12)

item\_att['AttributeEndDate'] = item\_att['AttributeEndDate']+timedelta(hours=12)

return (item\_att)

def extractCustomer(path,filename):

"""

This function extracts the sales transaction data from Customer file to python

It returns the Customer data frame

"""

cust\_cols = {"LoyaltyCardNum":np.int64,

"HouseholdNum":np.int64,

"MemberFavStore":np.int64,

"City":str,

"State":str,

"ZipCode":str,

"ExtraCol":str

}

cust\_list = pd.read\_csv(path + filename,

sep = '|',

header = None,

encoding='latin\_1',

quoting=3,

names=cust\_cols.keys())

cust\_list.isna().sum()

cust\_list.fillna({'LoyaltyCardNum':-999,

'HouseholdNum':-999,

'MemberFavStore':-999},inplace=True)

cust\_list = cust\_list.astype(cust\_cols)

cust\_list = cust\_list.drop(['ExtraCol'],axis = 1)

return (cust\_list)

def extractItemList(path,filename):

item\_cols = {"UPC":str,

"ItemID":str,

"Status":np.int64,

"LongDes":str,

"ShortDes":str,

"ClassCode":np.int64,

"ClassDes":str,

"CategoryCode":np.int64,

"CategoryDes":str,

"FamilyCode":np.int64,

"FamilyDes":str,

"DepartmentCode":np.int64,

"StoreBrand":str,

"ExtraDes":str,

"ExtraCol":str}

item\_list = pd.read\_csv(path+filename,

sep = '|',

header = None,

encoding='latin1',

names=item\_cols.keys(),

dtype=item\_cols,

quoting=3)

item\_list.replace({'Status':{0:'Invalid Item',

1:'Active',

2:'Suspended',

3:'Deleted'}},inplace=True)

item\_list['UPC'].fillna('-999',inplace=True)

item\_list = item\_list.drop(['ExtraCol'],axis = 1)

item\_list.dtypes

return(item\_list)

def connectToMongo(hostname,port):

"""

This function estalishes connection to Mongo DB and returns connection object

"""

mongoConnect = MongoClient(host='127.0.0.1',port=27017)

return mongoConnect

def insertIntoMongoDF(database,collection,mongo\_conn,df):

"""

This function insert the data into appropriate collection in mongo DB

"""

db = mongo\_conn.get\_database(database)

db.get\_collection(collection).drop()

db.create\_collection(collection)

db.get\_collection(collection).insert\_many(df.to\_dict(orient='record'))

def insertIntoMongoDict(database,collection,mongo\_conn,records):

"""

This function insert the data into appropriate collection in mongo DB

"""

db = mongo\_conn.get\_database(database)

db.get\_collection(collection).drop()

db.create\_collection(collection)

db.get\_collection(collection).insert\_many(records)

def loadHierarchy(df,conn,database,collection):

hierRecords=[{'UPC':df.loc[i,'UPC'],

'ItemID':df.loc[i,'ItemID'],

'DepartmentCode':int(df.loc[i,'DepartmentCode']),

'Family':{'FamilyCode':int(df.loc[i,'FamilyCode']),

'FamilyDesc':df.loc[i,'FamilyDes'],

'Category':{'CategoryCode':int(df.loc[i,'CategoryCode']),

'CategoryDesc':df.loc[i,'CategoryDes'],

'Class':{'ClassCode':int(df.loc[i,'ClassCode']),

'ClassDesc':df.loc[i,'ClassDes']}}}} for i in range(0,len(df))]

conn.get\_database(database).get\_collection(collection).drop()

conn.get\_database(database).create\_collection(collection)

conn.get\_database(database).get\_collection(collection).insert\_many(hierRecords)

def extractStoreLoc(path,filename):

store\_loc\_cols={'StoreNum':np.int64,

'StoreName':str,

'ActiveFlag':str,

'AddressLine1':str,

'City':str,

'StateCode':str,

'ZipCode':str,

'SqFoot':np.int64,

'Region':str,

'ClusterName':str,

'ExtraCol':str

}

storeLoc= pd.read\_csv(path+filename,

skiprows=2,

sep='|',

header=None,

names=store\_loc\_cols.keys())

nullRecordInd = storeLoc[storeLoc.isna().sum(axis=1)>=storeLoc.shape[1]-1].index.values

storeLoc.drop(index=nullRecordInd,axis=0,inplace=True)

storeLoc = storeLoc.astype(store\_loc\_cols)

storeLoc.drop(columns=['ExtraCol'],axis=1,inplace=True)

return(storeLoc)

def extractScrapedStore(path):

files = os.listdir(scrape\_path)

storeScraped=pd.DataFrame([])

temp\_cols = {'StoreName':str,

'StoreId':np.int64,

'LocationName':str,

'State':str,

'ZipCode':str,

'ServiceName':str,

'ServiceValue':str}

for file in files:

temp = pd.read\_csv(scrape\_path+file,

header=None)

for i in range(0,len(temp.columns)):

for j in range(0,len(temp)):

temp.loc[j,i]=temp.loc[j,i][temp.loc[j,i].find(':')+1:]

temp.columns = temp\_cols.keys()

temp = temp.astype(temp\_cols)

storeScraped=pd.concat([storeScraped,temp],ignore\_index=True)

storeScraped = storeScraped.astype(temp\_cols)

storeScraped.loc[~storeScraped['ServiceValue'].str.lower().isin(['true','false']),'ServiceValue'] = np.nan

storeRecords = []

for i in storeScraped.StoreId.unique():

df\_subset = storeScraped[storeScraped.StoreId==i]

df\_pivot = pd.pivot\_table(data=df\_subset,

aggfunc=lambda x: x,

columns='ServiceName',

values='ServiceValue',

index=['StoreId','StoreName','LocationName','State','ZipCode'])

df\_pivot.reset\_index(inplace=True)

rec\_service = df\_pivot.iloc[:,5:].to\_dict(orient='record')[0]

rec = df\_pivot.iloc[:,:5].to\_dict(orient='record')[0]

rec['Service']=rec\_service

storeRecords.append(rec)

return(storeRecords)

if \_\_name\_\_ == '\_\_main\_\_':

PATH = 'C:/Users/Universe/Desktop/DataScience/Spring 2018/BI/Project/dataFiles/'

scrape\_path = 'C:/Users/Universe/Desktop/DataScience/Spring 2018/BI/Project/dataFiles/scraping/'

SalesFile = 'sls\_dtl.txt'

ItemAttrFile = 'Item\_Attr.txt'

CustFile = 'customer\_List.txt'

ItemListFile = 'Item\_List.txt'

StoreLocFile = 'store\_list.txt'

SalesTrxClean = cleanSalesTrx(PATH,SalesFile)

salesDF = extractSalesTrx(SalesTrxClean)

itemAttrDF = extractItemAttr(PATH,ItemAttrFile)

custDF = extractCustomer(PATH,CustFile)

itemListDF = extractItemList(PATH,ItemListFile)

storeLocDF = extractStoreLoc(PATH,StoreLocFile)

DF = extractScrapedStore(scrape\_path)

conn\_obj = connectToMongo(hostname='127.0.0.1',port=27017)

insertIntoMongoDF('BIProject','SalesTrx',conn\_obj,salesDF)

insertIntoMongoDF('BIProject','ItemAttribute',conn\_obj,itemAttrDF)

insertIntoMongoDF('BIProject','Customer',conn\_obj,custDF)

insertIntoMongoDF('BIProject','ItemList',conn\_obj,itemListDF)

loadHierarchy(itemListDF,conn\_obj,'BIProject','ItemHierarchy')

insertIntoMongoDF('BIProject','StoreLocation',conn\_obj,storeLocDF)

insertIntoMongoDict('BIProject','StoreScraped',conn\_obj,DF)

**CODES for FTP**

import ftplib

import pandas as pd

#ftp = FTP("ftp://192.168.0.32/", "FTP\_user", "rudas4993")

ftp = ftplib.FTP('192.168.0.32')

ftp.login('FTP\_user', 'rudas4993')

# grt file from FTP and store in local file

def getfile():

filename = 'item\_list.txt'

localfile = open('C:/Users/rudas/Desktop/abc/filename','wb')

ftp.retrbinary('RETR ' + filename, localfile.write, 1024)

ftp.quit()

localfile.close()

# Load the local file

def load\_file():

dup\_list = pd.read\_csv('C:///Users///rudas///Desktop///abc///filename',sep = '|',

header = None,encoding='latin1')

col\_names = ["UPC","Status","RootID","LongDes","ShortDes","ClassCode","ClassDes","CategoryCode","CategoryDes",

"FamilyCode","FamilyDes","DepartmentCode","DepartmentDes","StoreBrand","ExtraDes","ec"]

dup\_list.columns = col\_names

dup\_list = dup\_list.drop(['ec'],axis = 1)

dup\_list.dtypes

# comparing two columns

def compare\_UPC():

new\_UPC = pd.DataFrame([])

for item in dup\_list['UPC']:

if item not in item\_list['UPC']:

new\_UPC.append(dup\_list)

getfile()

load\_file()

compare\_UPC()

## APPENDIX C – DIAGRAMS

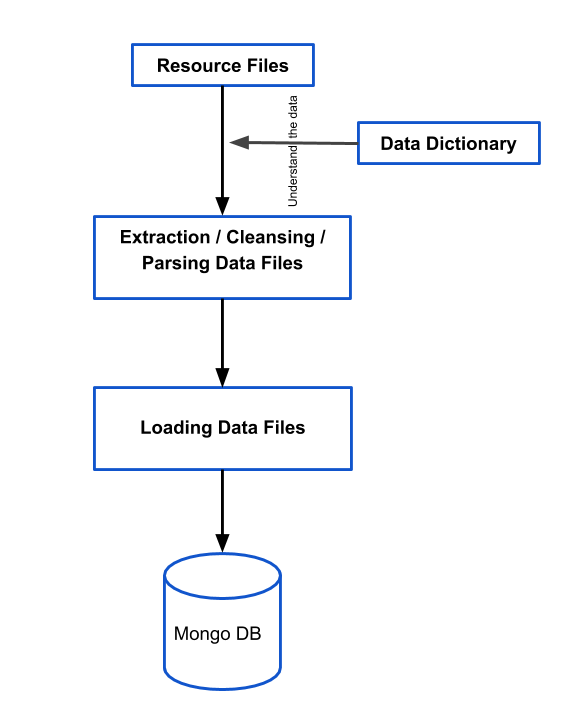


Figure 1 Diagram of First Task: ETL Process of Flat Data

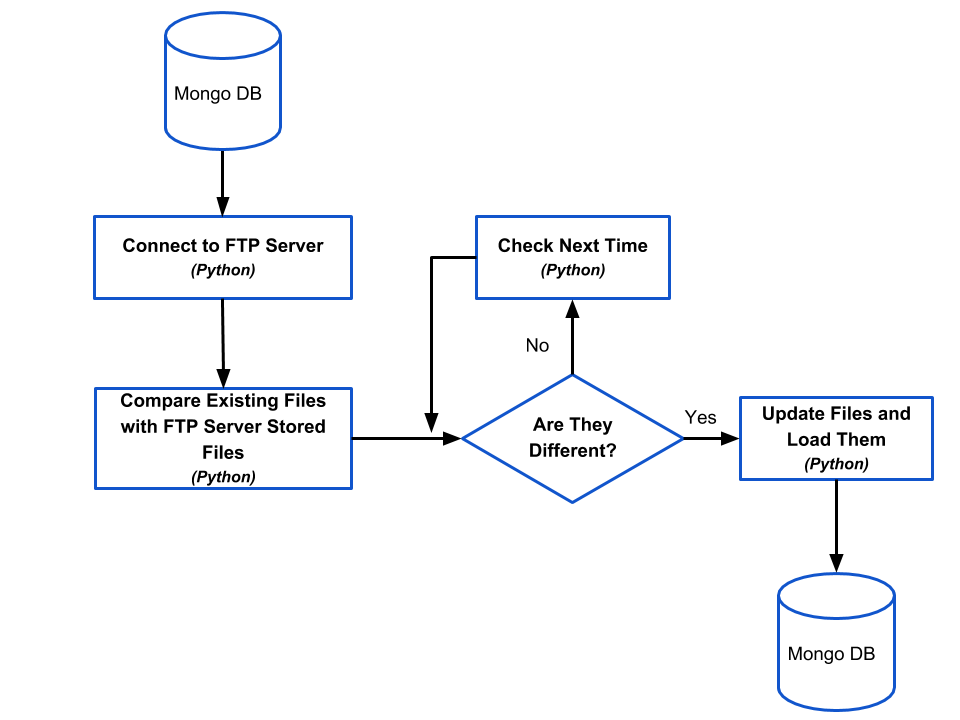


Figure 2 Diagram of Second Task: Updating Data File from FTP Server