EDA for Pro Finance Data Analysts

A Comprehensive Guide to Exploratory Data Analysis with Real-Life Finance Statements such as SEC Filings.

TODO: Download PDF version of this notebook

TODO: Video Tutorials


```
Author: Amit Shukla
```

https://github.com/AmitXShukla

https://x.com/ashuklax

https://youtube.com/@Amit.Shukla

by the end of this blog, you will learn techniques to

- Data Discovery using Pandas 2.0
- Create Data ERD diagram with animation (using manim)
- Data Visualization using Matplotlib
- Data Visualization using PlotLy
- Data Visualization using Seaborn
- Analyze Distributions
- Spot Anomalies
- Test Hypothesis
- Data Patterns
- Check Assumptions
- Create Interactive Visualizations
- what-if Analysis
- would, could, should
- Time Travel on Time Series Data
- Linear Regression, Auto Regression, SARIMA
- SVM
- Neural networks
- Graph Computing

Introduction

I'm Amit Shukla, and I specialize in training neural networks for finance supply chain analysis, enabling them to identify data patterns and make accurate predictions. During the challenges posed by the COVID-19 pandemic, I successfully trained GL and Supply Chain neural networks to anticipate supply chain shortages. The valuable insights gained from this effort have significantly influenced the content of this tutorial series.

Objective:

By delving into this powerful tool, we will master the fundamental techniques of using Exploratory Data Analysis. This knowledge is crucial in preparing finance and supply chain data for advanced analytics, visualization, and predictive modeling using neural networks and machine learning.

Subject

It's important to note that this particular series will concentrate solely on `Exploratory Data Analysis`.

Following

However, in future installments, we will explore Data Analytics and delve into the realm of machine learning for predictive analytics. Thank you for joining me, and I'm excited to embark on this educational journey together.

Let's get started.

Table of content

- Time Travel on Time Series Data
- Linear Regression, Auto Regression, SARIMA
- SVM
- Neural networks
- Graph Computing

what is EDA

EDA is often characterized as a tool for data analysts to discover patterns, spot anomalies, test a hypothesis, or check assumptions.

Installation

In [ ]:
```
# !pip install polars pandas numpy matplotlib seaborn tqdm

# import platform;
# print(platform.processor())
```

Technical & Fundamental Analysis

what is General Ledger

GL serves as core of any Financial Management system.

It's objective is to keep detail and summary accounting information and produce numerous financial reports for your organization. Typical, you will hear Cash Flow, Income and Balance Sheet statements as SEC filings as financial reports indicating Organizations financial growth. In general, accountants, statistical analysts strongly feel that financial reports from General Ledger are true indicator of organizations growth.

Technical Analysis

is the process of forecasting future Organization growth or stock prices based on studying (using advance charting and applying mathematical formulas) past stock prices and trading volume.

Technical analysis strongly believes that at any given point of time, stock price and trading volume reflects it current value and charting accurately captures all factors which can cause upwards or downwards stock prices movement.

Fundamental Analysis

is the process of forecasting future Organization growth or stock prices based on studying company Financial Statements like Finance Ledger, Balance Sheet, Income, Cash Flow Statements.

Techno-Fundamental Analysis

In this notebook, I am proposing to use 3rd type of analysis. With the use of Machine Learning, One can apply Statistical Analysis, ML algorithms to apply statistical data associations to Ledger, Sub-Ledger (accounting entries) statements along with Company Stock prices and trading volume.

`Techno-fundamental analysis is not new, however, its seen very difficult because its requires big data and large and fast computations for large data sets and great assets for Statistical programming.`

Finance data model

A finance data model is a comprehensive and structured framework used to represent and organize financial information within an organization.

It serves as the blueprint for how financial data is collected, stored, processed, and analyzed, ensuring accuracy, consistency, and efficiency in managing financial operations.

The model defines the relationships between various financial entities such as assets, liabilities, revenues, expenses, and equity, enabling financial professionals to gain insights into the company's financial health, performance, and risk exposure.

It typically encompasses multiple dimensions, including time, currency, and geographical locations, chart of accounts, departments / cost centers, fiscal years and reporting accounting periods providing a holistic view of the organization's financial landscape.

A well-designed finance data model is critical for generating accurate financial reports, facilitating financial planning and forecasting, and supporting strategic decision-making at all levels of the business.

As stated above, since our objective is learn Data Science operations on Finance and Supply chain dataset, we will focus on creating few real life examples which are similar to Finance and Supply chain.

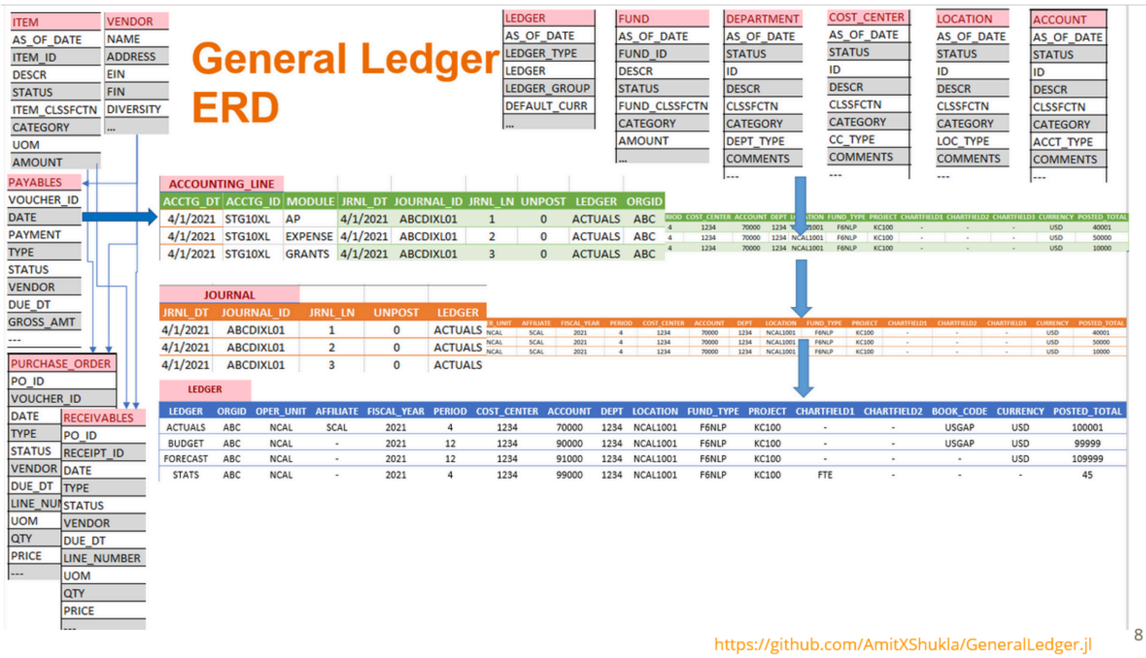
For more information, please learn more about Finance and Supply chain ERP data.

Objective of following section is to understand ERP GL like data.

A sample of data structure and ERD relationship diagram can be seen in this diagram below.

The ERD presented below depicts the data set that serves as the foundation for generating Finance Statements.

`Finance ER Diagram`



Supply chain data model

A supply chain data model is a structured representation of the various elements and interactions within a supply chain network.

It encompasses critical components such as customers, orders, receipts, products, invoices, vouchers, and ship-to locations.

Customers form the foundation of the supply chain, as they drive demand for products. Orders and receipts represent the flow of goods and services, capturing the movement of inventory throughout the supply chain.

The product entity accounts for the diverse range of items being handled, from raw materials to finished goods.

Invoices and vouchers track financial transactions, ensuring transparent and accurate billing processes.

Ship-to locations specify the destinations of goods during the distribution process.

By establishing relationships and attributes between these elements, the supply chain data model aids in optimizing inventory management, forecasting demand, enhancing order fulfillment, and ultimately, improving overall operational efficiency within the supply chain ecosystem.

`Supply Chain ER Diagram`

ERD Diagram

Loading Finance, Supply chain and Stock Prices Data

Financial & stock prices data

as stated in earlier sections, we will use real life examples (Tesla Inc.) in our analysis. We will first download real life Finance statement data and then later we will derive/create synthetic data from These Finance statements for our exploratory data analysis purpose.

download Financial Statement data to support Fundamental analysis https://ir.tesla.com/sec-filings

download Stock market data to support Technical analysis https://finance.yahoo.com/quote/TSLA/history/

In [ ]:

```
import os, shutil
os.listdir("../SampleData")

# we will work through TSLA**.csv files in below sections
# you will see that these files contain real data
# and real data is very messy in real world,
# so we will spend much time in data cleansing and transformation
```

Out[ ]:

```
['Amit_TestQRcode.png',
 'ER_Flow.png',
 'Process_Flow.png',
 'sampleData.csv',
 'The Ultimate Guide to Data Wrangling with Python - Rust Polars Data Frame.pdf',
 'TSLA.csv',
 'TSLA_Fin_Statements.xlsx']
```

load SEC Filings and Other Finance market data

please note that we have extensively discussed this topic in past tutorials. Below are couple of links to video tutorials that can assist you in extracting information from a specific page, like a website contraining links to download SEC filings and Financial statemetns. You can utilize the following links to download data and streamline the downloading process.

Please be aware that web scrapping may not be advisable due to the sensitive nature of Finance Statements. Ensure you obtain appropriate approvals and rely on authentic APIs when gathering such data.

Download csv, pdf, xls data files from web pages using Open AI ChatGPT Python code

In [ ]:

```
# this code is used to download Finance Statements data from Edgar or SEC filing we

import requests
from io import BytesIO
from zipfile import ZipFile, BadZipFile
from pathlib import Path
from tqdm import tqdm # show a progress meter, wrap any iterable in tqdm
import pandas as pd

# define URLs
download_URL = "https://www.sec.gov/files/dera/data/financial-statement-and-notes-d
user_agent = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (K

period = 2015
qtr = 1
url = f"{download_URL}{period}q{qtr}_notes.zip"
target = Path("../downloads/SEC")
response = requests.get(url, headers={"User-Agent": user_agent}).content
with ZipFile(BytesIO(response)) as zip_file:
    for file in zip_file.namelist():
        local_file = target
        if local_file.exists():
            continue
        with local_file.open("wb") as output:
                for line in zip_file.open(file).readlines():
                    output.write(line)

import os
print(os.listdir("../downloads")) # show downloaded file
```

Data Discovery using Pandas 2.0

In the previous section, we downloaded data locally.

In this section, we will learn techniques for reading, transformation and understanding data using Pandas framework.

Please click here if you are interested to learn more about using `Rust Polars DataFrame`.

In [ ]:

```
# !pip install polars pandas numpy matplotlib seaborn openpyxl
```

```
# I assume, you have already created an EDA virtual environment and installed these
```

Importing and Exporting data

The Pandas library offers functionalities for importing and exporting data in various formats. The syntax for most of these methods, such as `read_csv`, `read_excel`, or `read_parquet`, is quiet similar. The same applies to writing data with methods like `to_csv`, `to_excel` and `to_parquet`.

There's no need to memorize all these methods as there are numerous ones, each with different options and parameters. It;s more beneficial to understand the method signatures and experiment with the options while working with data.

In [ ]:

```
import os
import pandas as pd
os.listdir("../SampleData/")
df1 = pd.read_csv("../SampleData/TSLA.csv")
df1.sample(5)

df_temp = pd.ExcelFile("../SampleData/TSLA_Fin_Statements.xlsx")
df_temp.sheet_names
# df21 = pd.read_excel("../SampleData/TSLA_Fin_Statements.xlsx", sheet_name="balanc
# df21.describe()
# df21.head(5)

# df22 = pd.read_excel("../SampleData/TSLA_Fin_Statements.xlsx", sheet_name="income
# df22.describe()
# df22.head(5)

df23 = pd.read_excel("../SampleData/TSLA_Fin_Statements.xlsx", sheet_name="cashflow
df23.describe()
df23.tail(5)

# alternatively, use ths syntex to read all excel sheets into data frame
# # Each Excel sheet in a Python dictionary
# workbook = pd.ExcelFile('../SampleData/TSLA_Fin_Statements.xlsx')
# dictionary = {}
# for sheet_name in workbook.sheet_names:
#     df = workbook.parse(sheet_name)
#     dictionary[sheet_name] = df
# Note: the parse() method takes many arguments
# dictionary.keys()

###########################
### write results to csv
###########################
# to_csv is used to write dataframes into csv
# df1.to_csv("../SampleData/TSLA_downloaded.csv")
```

Out[ ]:

```
dict_keys(['balancesheet', 'income', 'cashflow'])
```

Data Structure

The core base data structure provided by Pandas is Series and DataFrame.

`Series` is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the `index`.

`DataFrame` is a two-dimensional labeled array capable of holding `Series` (s)/columns or sometime referred as `vectors` of any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the `index`.

In [ ]:

```
import numpy as np
s = pd.Series(np.random.randn(5))
# s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])

print(s)
print(s.index)

# similarly a DataFrame is made up of series/vectors as columns
d = {
    "one": pd.Series([1.0, 2.0, 3.0], index=["a", "b", "c"]),
    "two": pd.Series([1.0, 2.0, 3.0, 4.0], index=["a", "b", "c", "d"]),
}
df = pd.DataFrame(d)
print(df)
print(df.index)

# --- get Index from Series and DataFrame
print(s.index)
print(df.columns)
print(df.index)
print("axes:")
print(df.axes)

# --- Index properties
print(df.index.is_monotonic_decreasing)
print(df.index.is_monotonic_increasing)
print(df.index.has_duplicates)
print(df.index.nlevels)

# --- Index methods
print(df.index.values)
print(df.index.value_counts)
print(df.index.tolist())
print(df.index.nunique())
print(df.index.min())
print(df.index.max())
```

```
0   -0.653677
1   -1.337561
2   -0.717682
3   -0.452272
4   -0.057651
dtype: float64
RangeIndex(start=0, stop=5, step=1)
   one  two
a  1.0  1.0
b  2.0  2.0
c  3.0  3.0
d  NaN  4.0
Index(['a', 'b', 'c', 'd'], dtype='object')
RangeIndex(start=0, stop=5, step=1)
Index(['one', 'two'], dtype='object')
Index(['a', 'b', 'c', 'd'], dtype='object')
axes:
[Index(['a', 'b', 'c', 'd'], dtype='object'), Index(['one', 'two'], dtype='object')]
False
True
False
1
['a' 'b' 'c' 'd']
<bound method IndexOpsMixin.value_counts of Index(['a', 'b', 'c', 'd'], dtype='objec
t')>
['a', 'b', 'c', 'd']
4
a
d
```

In [ ]:

```
# example series and Data frame

import random
from datetime import datetime
import pandas as pd

descr = pd.Series(["Boston","New York","Philadelphia","Cleveland","Richmond",
                     "Atlanta","Chicago","St. Louis","Minneapolis","Kansas City",
                     "Dallas","San Francisco"], name = "DESCRIPTION")
print(descr)

location = pd.DataFrame({
    "ID":  list(range(11, 23)),
    "AS_OF_DATE" : datetime(2022, 1, 1),
    "DESCRIPTION" : descr,
    "REGION": ["Region A","Region B","Region C","Region D"] * 3,
    "TYPE" : "Physical",
    "CATEGORY" : ["Ship","Recv","Mfg"] * 4
})
print(location.shape)
location.head(5)
```

```
0           Boston
1         New York
2     Philadelphia
3        Cleveland
4         Richmond
5          Atlanta
6          Chicago
7        St. Louis
8      Minneapolis
9      Kansas City
10          Dallas
11   San Francisco
Name: DESCRIPTION, dtype: object
(12, 6)
```

Out[ ]:

	ID	AS_OF_DATE	DESCRIPTION	REGION	TYPE	CATEGORY
0	11	2022-01-01	Boston	Region A	Physical	Ship
1	12	2022-01-01	New York	Region B	Physical	Recv
2	13	2022-01-01	Philadelphia	Region C	Physical	Mfg
3	14	2022-01-01	Cleveland	Region D	Physical	Ship
4	15	2022-01-01	Richmond	Region A	Physical	Recv

In [ ]:

```
# use this script to create synthetic Finance, Supply chain dataset

import pandas as pd
import os

dirPath = "../../../downloads/" # directory where sample csv are generated
sampleSize = 100_000 # generate 100k sample rows

print(os.listdir(dirPath))

# Creating DataFrame from a dict or a collection of dicts.
# let's create a more sophisticated DataFrame
# in real world, Organization maintain dozens of record structure to store
# different type of locations, like ShipTo Location, Receiving,
# Mailing, Corp. office, head office,
# field office etc. etc.

########################
## LOCATION DataFrame ##
########################
import random
from datetime import datetime

location = pd.DataFrame({
    "ID":  list(range(11, 23)),
    "AS_OF_DATE" : datetime(2022, 1, 1),
    "DESCRIPTION" : ["Boston","New York","Philadelphia","Cleveland","Richmond",
                     "Atlanta","Chicago","St. Louis","Minneapolis","Kansas City",
                     "Dallas","San Francisco"],
    "REGION": ["Region A","Region B","Region C","Region D"] * 3,
```

```
    "TYPE" : "Physical",
    "CATEGORY" : ["Ship","Recv","Mfg"] * 4
})
location.head()

########################
## ACCOUNTS DataFrame ##
########################

accounts = pd.DataFrame({
    "ID":  list(range(10000, 45000, 1000)),
    "AS_OF_DATE" : datetime(2022, 1, 1),
    "DESCRIPTION" : ["Operating Expenses","Non Operating Expenses","Assets",
                     "Liabilities","Net worth accounts", "Statistical Accounts",
                     "Revenue"] * 5,
    "REGION": ["Region A","Region B","Region C","Region D", "Region E"] * 7,
    "TYPE" : ["E","E","A","L","N","S","R"] * 5,
    "STATUS" : "Active",
    "CLASSIFICATION" : ["OPERATING_EXPENSES","NON-OPERATING_EXPENSES",
                        "ASSETS","LIABILITIES","NET_WORTH","STATISTICS",
                        "REVENUE"] * 5,
    "CATEGORY" : [
                "Travel","Payroll","non-Payroll","Allowance","Cash",
                "Facility","Supply","Services","Investment","Misc.",
                "Depreciation","Gain","Service","Retired","Fault.",
                "Receipt","Accrual","Return","Credit","ROI",
                "Cash","Funds","Invest","Transfer","Roll-over",
                "FTE","Members","Non_Members","Temp","Contractors",
                "Sales","Merchant","Service","Consulting","Subscriptions"
        ],
})
accounts.head()

##########################
## DEPARTMENT DataFrame ##
##########################

dept = pd.DataFrame({
    "ID":  list(range(1000, 2500, 100)),
    "AS_OF_DATE" : datetime(2022, 1, 1),
    "DESCRIPTION" : ["Sales & Marketing","Human Resource",
                     "Information Technology","Business leaders","other temp"] * 3,
    "REGION": ["Region A","Region B","Region C"] * 5,
    "STATUS" : "Active",
    "CLASSIFICATION" : ["SALES","HR", "IT","BUSINESS","OTHERS"] * 3,
    "TYPE" : ["S","H","I","B","O"] * 3,
    "CATEGORY" : ["sales","human_resource","IT_Staff","business","others"] * 3,
})
dept.head()

######################
## LEDGER DataFrame ##
######################

org = "ABC Inc."
ledger_type = "ACTUALS" # BUDGET, STATS are other Ledger types
```

```
fiscal_year_from = 2020
fiscal_year_to = 2023
random.seed(123)

ledger = pd.DataFrame({
        "LEDGER" : ledger_type,
        "ORG" : org,
        "FISCAL_YEAR": random.choices(list(range(fiscal_year_from,
                                          fiscal_year_to+1, 1)),k=sampleSize),
        "PERIOD": random.choices(list(range(1, 12+1, 1)),k=sampleSize),
        "ACCOUNT" : random.choices(accounts["ID"], k=sampleSize),
        "DEPT" : random.choices(dept["ID"], k=sampleSize),
        "LOCATION" : random.choices(location["ID"], k=sampleSize),
        "POSTED_TOTAL": random.sample(range(1000000), sampleSize)
})
ledger.sample(5)

ledger_type = "BUDGET" # ACTUALS, STATS are other Ledger types

ledgerBudget = pd.DataFrame({
        "LEDGER" : ledger_type,
        "ORG" : org,
        "FISCAL_YEAR": random.choices(list(range(fiscal_year_from, fiscal_year_to+1
                                    ,k=sampleSize),
        "PERIOD": random.choices(list(range(1, 12+1, 1)),k=sampleSize),
        "ACCOUNT" : random.choices(accounts["ID"], k=sampleSize),
        "DEPT" : random.choices(dept["ID"], k=sampleSize),
        "LOCATION" : random.choices(location["ID"], k=sampleSize),
        "POSTED_TOTAL": random.sample(range(1000000), sampleSize)
})
ledgerBudget.sample(5)

#########################################
# combined Ledger for Actuals and Budget
#########################################
dfLedger = pd.concat([ledger, ledgerBudget])
dfLedger.sample(5)

location.to_csv(f"{dirPath}location.csv")
dept.to_csv(f"{dirPath}dept.csv")
accounts.to_csv(f"{dirPath}accounts.csv")
dfLedger.to_csv(f"{dirPath}ledger.csv")

print(os.listdir(dirPath))
dfLedger.shape
```

```
['accounts.csv', 'dept.csv', 'earth.jpg', 'ledger.csv', 'ledger.json', 'ledger.parqu
et', 'location.csv']
['accounts.csv', 'dept.csv', 'earth.jpg', 'ledger.csv', 'ledger.json', 'ledger.parqu
et', 'location.csv']
```

Out[ ]: (200000, 8)

Data Exploration using Pandas basics functionalities & Indexing

In [ ]:
```
# for now, we will start with simple example
# as you can see, TSLA.csv which has historical stock prices and is easier to read
# once we lean basics of indexing/viewing functionalities, we will use these techni
# to standardize more complex data as shown in TSLA Financial Statement file

##########################
## quick glance through ##
##########################
dfLedger.head(3)
dfLedger.tail(3)
dfLedger.sample
dfLedger.ndim
dfLedger.axes
dfLedger.size
dfLedger.shape
dfLedger.index
dfLedger.index.array
dfLedger.columns
dfLedger.dtypes
dfLedger.values

# # DataFrame iteration methods
# dfLedger.iteritems()# (col-index, Series) pairs # NA - will fail
dfLedger.iterrows() # (row-index, Series) pairs

dfLedger["LEDGER"].value_counts() # The value_counts() Series method computes a his
data = {"a": [1, 2, 3, 4], "b": ["x", "x", "y", "y"]}
frame = pd.DataFrame(data)
frame.value_counts()

df1 = pd.read_csv("../SampleData/TSLA.csv")
df1["Date"].__len__()
df1["Date"].values
df1["Date"].to_numpy(dtype=object)
df1["Date"].to_numpy(dtype="datetime64[ns]")
df1.to_numpy()
df1["Close"].array
df1.describe()
df1["Close"].describe()

####################################
## data Descriptive statistics #####
## count, sum, mean, median, min,
####################################
dfLedger.info()
dfLedger.describe()
df1["High"].count()
# count # Number of non-NA observations
# sum # Sum of values
# mean # Mean of values
```

```
# median # Arithmetic median of values
# min # Minimum
# max # Maximum
# idxmin() and idxmax()
# mode # Mode
# abs # Absolute Value
# prod # Product of values
# std # Bessel-corrected sample standard deviation
# var # Unbiased variance
# sem # Standard error of the mean
# skew # Sample skewness (3rd moment)
# kurt # Sample kurtosis (4th moment)
# quantile # Sample quantile (value at %)
# cumsum # Cumulative sum
# cumprod # Cumulative product
# cummax # Cumulative maximum
# cummin # Cumulative minimum

# df1["High"].cummin()

# element-wise methods
# df1['High'].isnull()
# df1['High'].notnull() # not isnull()
# df1['High'].astype(float)
# df1['High'].round(decimals=0)
# df1['High'].diff(periods=1)
# df1['High'].shift(periods=1)
# df1['Date'].to_datetime()
# df1['High'].fillna(0) # replace NaN w 0
# df1['High'].cumsum()
# df1['High'].cumprod()
# df1['High'].pct_change(periods=4)
# df1['High'].rolling_sum(periods=4, window=4)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 200000 entries, 0 to 99999
Data columns (total 8 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   LEDGER        200000 non-null  object
 1   ORG           200000 non-null  object
 2   FISCAL_YEAR   200000 non-null  int64
 3   PERIOD        200000 non-null  int64
 4   ACCOUNT       200000 non-null  int64
 5   DEPT          200000 non-null  int64
 6   LOCATION      200000 non-null  int64
 7   POSTED_TOTAL  200000 non-null  int64
dtypes: int64(6), object(2)
memory usage: 13.7+ MB
```

```
Out[ ]: 251
```

Data selection and indexing

In [ ]:
```
##########################
## working with columns ##
##########################

print(dfLedger.columns)
print(dfLedger.columns.tolist())
dfLedger.rename(columns={'FISCAL_YEAR':'FY', 'ORG':'ORGANIZATION'}, inplace=True)
print(dfLedger.columns.tolist())
dfLedger.rename(columns={'FY':'FISCAL_YEAR','ORGANIZATION':'ORG'}, inplace=True)
print(dfLedger.columns.tolist())

# Selecting columns
dfLedger['LEDGER'] # returns a series datatype
dfLedger[['LEDGER']] # returns a data frame datatype
dfLedger[['LEDGER', 'FISCAL_YEAR']] # returns a data frame datatype

dfLedger[dfLedger.columns[0]] # select column by position
dfLedger[dfLedger.columns[[0,1,2]]] # select columns by position

dfLedger["FISCAL_YEAR"]
dfLedger.FISCAL_YEAR
dfLedger["FISCAL_YEAR"].value_counts()
# add a new column
dfLedger["FYP"] = dfLedger["FISCAL_YEAR"].astype(str) + "-" + dfLedger["PERIOD"].as
dfLedger.FYP
# dfLedger.pop("FYP")
dfLedger.drop("FYP", axis=1, inplace=True)
# del dfLedger['FYP']

# Vectorised column calculations
# this is very useful in feature normalization | standardization
dfLedger['ranked']=dfLedger['POSTED_TOTAL']*100000/sum(dfLedger.POSTED_TOTAL)
max(dfLedger.ranked)
min(dfLedger.ranked)

# other numpy mathematical functions to columns
import numpy as np
np.seterr(divide = 'ignore') # ignore log func divide by zero warning
dfLedger['new_ranked'] = np.log(dfLedger['POSTED_TOTAL'])
dfLedger['new_ranked'] = np.round(dfLedger['new_ranked'],2)

del dfLedger["ranked"]
del dfLedger["new_ranked"]

# Columns value set based on criteria
dfLedger['POSTED_TOTAL']=dfLedger['POSTED_TOTAL'].where(dfLedger['POSTED_TOTAL']>0,
dfLedger
```

```
Index(['LEDGER', 'ORG', 'FISCAL_YEAR', 'PERIOD', 'ACCOUNT', 'DEPT', 'LOCATION',
       'POSTED_TOTAL'],
      dtype='object')
['LEDGER', 'ORG', 'FISCAL_YEAR', 'PERIOD', 'ACCOUNT', 'DEPT', 'LOCATION', 'POSTED_TO
TAL']
['LEDGER', 'ORGANIZATION', 'FY', 'PERIOD', 'ACCOUNT', 'DEPT', 'LOCATION', 'POSTED_TO
TAL']
['LEDGER', 'ORG', 'FISCAL_YEAR', 'PERIOD', 'ACCOUNT', 'DEPT', 'LOCATION', 'POSTED_TO
TAL']
```

Out[ ]:

	LEDGER	ORG	FISCAL_YEAR	PERIOD	ACCOUNT	DEPT	LOCATION	POSTED_TOTA
0	ACTUALS	ABC Inc.	2020	10	12000	2400	22	75395
1	ACTUALS	ABC Inc.	2020	1	10000	1900	14	82690
2	ACTUALS	ABC Inc.	2021	12	21000	1700	17	45457
3	ACTUALS	ABC Inc.	2020	3	34000	1300	12	33498
4	ACTUALS	ABC Inc.	2023	9	14000	2100	11	29081
...	...	...	...	...	...	...	...	
99995	BUDGET	ABC Inc.	2023	7	16000	1900	14	58725
99996	BUDGET	ABC Inc.	2023	1	26000	1000	17	66383
99997	BUDGET	ABC Inc.	2022	5	20000	2400	17	6543
99998	BUDGET	ABC Inc.	2020	8	11000	1900	17	96025
99999	BUDGET	ABC Inc.	2022	7	42000	2300	11	4115

200000 rows × 8 columns

In [ ]:

```python
# Selecting columns with .loc, .iloc and .ix
dfLedger.loc[:, 'LEDGER':'PERIOD'] # inclusive
dfLedger.iloc[:, 0:4] # exclusive
# Get the integer position of a column index label
dfLedger.columns.get_loc('POSTED_TOTAL')

# Selecting scalars with .at, .iat
dfLedger.at[4, 'POSTED_TOTAL'] # inclusive
dfLedger.iat[4, 7] # exclusive

# filter selections
```

```
dfLedger.filter(items=['ORG']) # by col
dfLedger.filter(like='D') # keep D in col
dfLedger.filter(regex='D') # regex in col

###############################
## working with rows & cells ##
###############################
dfLedger.index.to_list()
# dfLedger.rename(index={'old':'new'}, inplace=True)

# if indexes are not alinged or you need to re-assign indexes
# dfLedger.reindex(index=range(len(dfLedger)), method="bfill")
# since we merged two dataframes earlier, re-index will not work
# until we re-index one of merged dataframes in previous section
dfLedger[dfLedger.index.duplicated()]
dfLedger.loc[5, "LEDGER"]
dfLedger.at[4, "POSTED_TOTAL"]
dfLedger.iat[4, dfLedger.columns.get_loc('POSTED_TOTAL')]
```

Out[ ]:
```
290813
```

Pandas to load data into DataFrames

in this section, we will load real life data from Data sources into data frame and in cases where detail data is reqiured, we will create synthetic data to simulate a real life exploratory data analysis.

In [ ]:
```
import os
dirPath = "../../../downloads/"
# print(os.listdir(dirPath))

# as you can see, there are many files,
# we will focus on loading only csv/xls/xlsx files for now

for filename in os.listdir(dirPath):
    if filename.endswith(".csv"):
        print("Eligible to read into DataFrame: ", filename)

import pandas as pd
dfAccounts = pd.read_csv(dirPath+"accounts.csv")
dfDept = pd.read_csv(dirPath+"dept.csv")
dfLocation = pd.read_csv(dirPath+"location.csv")
dfLedger = pd.read_csv(dirPath+"ledger.csv")

print(dfAccounts.shape, dfDept.shape, dfLocation.shape, dfLedger.shape)
dfLedger.sample(10)
```

```
Eligible to read into DataFrame:  accounts.csv
Eligible to read into DataFrame:  dept.csv
Eligible to read into DataFrame:  ledger.csv
Eligible to read into DataFrame:  location.csv
(35, 8) (15, 8) (12, 6) (200000, 8)
```

Out[ ]:

	LEDGER	ORG	FISCAL_YEAR	PERIOD	ACCOUNT	DEPT	LOCATION	POSTED_TOT
16322	ACTUALS	ABC Inc.	2022	5	39000	2400	20	9197
41713	ACTUALS	ABC Inc.	2021	10	27000	1800	22	2824
94363	ACTUALS	ABC Inc.	2020	5	44000	2000	16	6223
179527	BUDGET	ABC Inc.	2020	7	35000	1300	21	7393
176177	BUDGET	ABC Inc.	2022	12	39000	1200	17	5734
37782	ACTUALS	ABC Inc.	2022	11	23000	1300	22	510
108868	BUDGET	ABC Inc.	2021	3	38000	1300	18	5002
112988	BUDGET	ABC Inc.	2022	7	17000	1700	16	1027
39437	ACTUALS	ABC Inc.	2021	11	15000	1000	13	9239
72200	ACTUALS	ABC Inc.	2023	2	42000	2400	22	4174

Creating Synthetic Data from real samples

PS: Just wanted to give you a heads up that in DSPy, a programming language we'll be using for LLMs and RAGs in Finance, there's a nifty tool called `Synthesizer` that can create synthetic data based on the input data you give it.

We'll be using this tool later on, but for now, we'll stick to creating synthetic data using simple scripts.

Below is a sample script which creates synthetic data.

This script aims to create a dataset with a purposeful bias during specific periods, fostering an environment for both exploratory data analysis and machine learning algorithms to detect and leverage these trends.

The objective is to enhance prediction accuracy. By introducing a certain bias, we avoid generating a random and ultimately unhelpful dataset. This method assumes real-life datasets inherently contain hidden patterns that can be revealed through careful analysis and prediction.

This exercise's main goal is to uncover such patterns, learn from the data, and train and predict based on these insights. For example, imagine a real-world dataset biased such that every second quarter has poor performance, while each subsequent quarter improves on the previous one.

In a real-world scenario, this dataset originates from production ERP (Enterprise Resource Planning) systems, which encompass various aspects of an organization, including HR, customer relationship management, supply chain, inventory, revenue, and financial operations. This dataset is derived from actual production ERP systems, which consolidate data from diverse functional areas like HR, customer management, supply chain, inventory, revenue, and finance within an organization.

In [ ]:

```
import pandas as pd
# df_temp = pd.ExcelFile("../SampleData/TSLA_Fin_Statements.xlsx")
# df_temp.sheet_names

df = pd.read_excel("../SampleData/TSLA_Fin_Statements.xlsx", sheet_name="balanceshe
dfAccountGrp = df[df[df.columns[0]].notnull()].iloc[:,0:1] # remove NaN
dfAccountGrp = dfAccountGrp[dfAccountGrp.duplicated(keep=False)] # remove dups
print(dfAccountGrp[dfAccountGrp.columns[0]].tolist())
# as you can see, it didn't do a good job at creating actual account names,
# another trick you can use is,
# run this list through a GPT LLM and retrieve list of real account names and elimi
# example prompt:
# Could you identify the values in the list that seem to resemble genuine account d

########################################
# here is the list created by ChatGPT ##
########################################
# Automoative sales
# Automoative leasing
# Energy generation and storage
# Services and other
# Net income
# Basic
# Diluted
# Exercises of conversion feature of convertible senior notes
# Issuance of common stock for equity incentive awards
# Stock-based compensation
# Distributions to noncontrolling interests
# Buy-outs of noncontrolling interests
# Other comprehensive loss
# Balance as of September 30, 2023
# Balance as of September 30, 2022
```

```
['(in millions, except per share data)', '(unaudited)', 'The accompanying notes are an integral part of these consolidated financial statements.', 'Table of Contents', 'Tesla, Inc.', '(in millions, except per share data)', '(unaudited)', 'Automotive sales', 'Automotive leasing', 'Energy generation and storage', 'Services and other', 'Automotive sales', 'Automotive leasing', 'Energy generation and storage', 'Services and other', 'Net income', 'Basic', 'Diluted', 'Basic', 'Diluted', 'The accompanying notes are an integral part of these consolidated financial statements.', 'Table of Contents', 'Tesla, Inc.', '(unaudited)', 'Net income', 'The accompanying notes are an integral part of these consolidated financial statements.', 'Table of Contents', 'Tesla, Inc.', '(in millions, except per share data)', '(unaudited)', 'Exercises of conversion feature of convertible senior notes', 'Issuance of common stock for equity incentive awards', 'Stock-based compensation', 'Distributions to noncontrolling interests', 'Buy-outs of noncontrolling interests', 'Net income', 'Other comprehensive loss', 'Balance as of September 30, 2023', 'Exercises of conversion feature of convertible senior notes', 'Issuance of common stock for equity incentive awards', 'Stock-based compensation', 'Distributions to noncontrolling interests', 'Buy-outs of noncontrolling interests', 'Net (loss) income', 'Other comprehensive loss', 'Balance as of September 30, 2023', 'Table of Contents', 'Exercises of conversion feature of convertible senior notes', 'Issuance of common stock for equity incentive awards', 'Stock-based compensation', 'Distributions to noncontrolling interests', 'Net income', 'Other comprehensive loss', 'Balance as of September 30, 2022', 'Exercises of conversion feature of convertible senior notes', 'Issuance of common stock for equity incentive awards', 'Stock-based compensation', 'Distributions to noncontrolling interests', 'Net (loss) income', 'Other comprehensive loss', 'Balance as of September 30, 2022', 'The accompanying notes are an integral part of these consolidated financial statements.']
```

In [ ]:

```
# use this script to create synthetic Finance, Supply chain dataset

import pandas as pd
import os

dirPath = "../../../downloads/" # directory where sample csv are generated
sampleSize = 100_000 # generate 100k sample rows

########################
## LOCATION DataFrame ##
########################
import random
from datetime import datetime

location = pd.DataFrame({
    "ID":  list(range(11, 23)),
    "AS_OF_DATE" : datetime(2022, 1, 1),
    "DESCRIPTION" : ["Boston","New York","Philadelphia","Cleveland","Richmond",
                     "Atlanta","Chicago","St. Louis","Minneapolis","Kansas City",
                     "Dallas","San Francisco"],
    "REGION": ["Region A","Region B","Region C","Region D"] * 3,
    "TYPE" : "Physical",
    "CATEGORY" : ["Ship","Recv","Mfg"] * 4
})
location.head()

########################
## ACCOUNTS DataFrame ##
########################
```

```
accounts = pd.DataFrame({
    "ID":  list(range(10000, 45000, 1000)),
    "AS_OF_DATE" : datetime(2022, 1, 1),
    "DESCRIPTION" : ["Operating Expenses","Non Operating Expenses","Assets",
                     "Liabilities","Net worth accounts", "Statistical Accounts",
                     "Revenue"] * 5,
    "REGION": ["Region A","Region B","Region C","Region D", "Region E"] * 7,
    "TYPE" : ["E","E","A","L","N","S","R"] * 5,
    "STATUS" : "Active",
    "CLASSIFICATION" : ["OPERATING_EXPENSES","NON-OPERATING_EXPENSES",
                        "ASSETS","LIABILITIES","NET_WORTH","STATISTICS",
                        "REVENUE"] * 5,
    "CATEGORY" : [
                "Travel","Payroll","non-Payroll","Allowance","Cash",
                "Facility","Supply","Services","Investment","Misc.",
                "Depreciation","Gain","Service","Retired","Fault.",
                "Receipt","Accrual","Return","Credit","ROI",
                "Cash","Funds","Invest","Transfer","Roll-over",
                "FTE","Members","Non_Members","Temp","Contractors",
                "Sales","Merchant","Service","Consulting","Subscriptions"
        ],
})
accounts.head()

##########################
## DEPARTMENT DataFrame ##
##########################

dept = pd.DataFrame({
    "ID":  list(range(1000, 2500, 100)),
    "AS_OF_DATE" : datetime(2022, 1, 1),
    "DESCRIPTION" : ["Sales & Marketing","Human Resource",
                     "Information Technology","Business leaders","other temp"] * 3,
    "REGION": ["Region A","Region B","Region C"] * 5,
    "STATUS" : "Active",
    "CLASSIFICATION" : ["SALES","HR", "IT","BUSINESS","OTHERS"] * 3,
    "TYPE" : ["S","H","I","B","O"] * 3,
    "CATEGORY" : ["sales","human_resource","IT_Staff","business","others"] * 3,
})
dept.head()

######################
## LEDGER DataFrame ##
######################

org = "ABC Inc."
ledger_type = "ACTUALS" # BUDGET, STATS are other Ledger types
fiscal_year_from = 2020
fiscal_year_to = 2023
random.seed(123)

ledger = pd.DataFrame({
        "LEDGER" : ledger_type,
        "ORG" : org,
        "FISCAL_YEAR": random.choices(list(range(fiscal_year_from,
```

```
                                          fiscal_year_to+1, 1)),k=sampleSize),
        "PERIOD": random.choices(list(range(1, 12+1, 1)),k=sampleSize),
        "ACCOUNT" : random.choices(accounts["ID"], k=sampleSize),
        "DEPT" : random.choices(dept["ID"], k=sampleSize),
        "LOCATION" : random.choices(location["ID"], k=sampleSize),
        "POSTED_TOTAL": random.sample(range(1000000), sampleSize)
})
ledger.sample(5)

ledger_type = "BUDGET" # ACTUALS, STATS are other Ledger types

ledgerBudget = pd.DataFrame({
        "LEDGER" : ledger_type,
        "ORG" : org,
        "FISCAL_YEAR": random.choices(list(range(fiscal_year_from, fiscal_year_to+1
                               ,k=sampleSize),
        "PERIOD": random.choices(list(range(1, 12+1, 1)),k=sampleSize),
        "ACCOUNT" : random.choices(accounts["ID"], k=sampleSize),
        "DEPT" : random.choices(dept["ID"], k=sampleSize),
        "LOCATION" : random.choices(location["ID"], k=sampleSize),
        "POSTED_TOTAL": random.sample(range(1000000), sampleSize)
})
ledgerBudget.sample(5)

#########################################
# combined Ledger for Actuals and Budget
#########################################
dfLedger = pd.concat([ledger, ledgerBudget])
dfLedger.sample(5)

# location.to_csv(f"{dirPath}location.csv")
# dept.to_csv(f"{dirPath}dept.csv")
# accounts.to_csv(f"{dirPath}accounts.csv")
# dfLedger.to_csv(f"{dirPath}ledger.csv")

print(os.listdir(dirPath))
dfLedger.shape
```

Joining data

merge, join, concatenate and compare

In [ ]:

```
####################################
## data sorting ####################
####################################
# iteraion
# sorting
# items
# Vectorized string methods

####################################
## data operations #################
## add(), sub(), mul(), div(), radd(), rsub()
####################################
```

```
####################################
## data copy, transformation #######
####################################
# Tablewise Function Application: pipe()
# Row or Column-wise Function Application: apply()
# Aggregation API: agg() and transform()
# Applying Elementwise Functions: map()

####################################
# data joins
####################################
# Aligning objects with each other with align
```

reshaping & pivot tables

handling missing data

query

computations

sparse data structure

timeseries, timedelta and date operations

chart visualization

table visualization

data analysis | statistics

Create Data ERD diagram with animation (using manim)

Data Visualization using Matplotlib

Data Visualization using PlotLy

Data Visualization using Seaborn

APPENDIX

TODO:

In the initial stage of Data Discovery, the primary step involves recognizing and establishing a dynamic repository that encompasses all accessible datasets. It is imperative to identify the relationships between these datasets before embarking on data transformation or analytics.

This phase is of utmost importance, as it entails creating an official diagram reminiscent of an Entity-Relationship Diagram (ERD). The crucial tasks include pinpointing data types and discerning the fields that contain valuable information. This not only aids in comprehending the data but also facilitates a deeper understanding of the business processes or the insights derived from these datasets.

In this section, we will delve into the Data Discovery phase. We will initiate the process by scrutinizing the available data and crafting an ERD that encapsulates the dataset structures.

Let's begin by examining the available dataset.

For now, we won't concern ourselves with its source, I'll provide the scripts used to generate it later.

Our goal is to simulate a real-world project scenario where analysts often receive unfamiliar datasets and initiate data exploration.

The following steps demonstrate this process, and we'll take it one step at a time to learn how to approach data discovery.

Keep in mind that there's no one-size-fits-all approach, it varies based on data types and quality.

Consider these steps as general guidelines. Let's begin.