```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'inventory-for-demand-forecasting:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F4703540%2F7989831%2Fbundle%2Farchive.zip%3FX-Goog-Algorithm%3

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
```

```
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

Start coding or generate with AI.

```
import numpy as np # linear algebra
import pandas as pd
```

```
import os
for dirname, _, filenames in os.walk('/content/train.csv.zip'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
import pandas as pd
```

```
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.graph_objects as go
```

```
train =  pd.read_csv("/kaggle/input/inventory-for-demand-forecasting/train.csv")
test =  pd.read_csv("/kaggle/input/inventory-for-demand-forecasting/test.csv")
sub =  pd.read_csv("/kaggle/input/inventory-for-demand-forecasting/sample_submission.csv")
```

```
train.shape
```

```
train.columns
```

```
test.columns
```

```
train.head()
```

```
test.head()
```

```
train.isnull().sum()
```

```
test.isnull().sum()
```

```
train.columns
```

```
train.dtypes
```

```
pd.to_datetime(train['date'])
```

```
train.dtypes
```

```
sd = train.date.iloc[0]
ed = train.date.iloc[-1]

print("Start Date:", sd)
print("End Date  :", ed)
```

```
sd = test.date.iloc[0]
ed = test.date.iloc[-1]

print("Start Date:", sd)
print("End Date  :", ed)
```

```
train.store.value_counts()


train.item.value_counts()


s = train.sales.value_counts()
s


train['date'] = pd.to_datetime(train['date'], format='%Y-%m-%d')

y_2013 = train.loc[(train['date'] >= '2012-12-31')
                     & (train['date'] <= '2013-12-31')]


monthvise= y_2013.groupby(y_2013['date'].dt.strftime('%B'))[['store','item','sales']]


y_2013[:365]


fig = px.line(y_2013[:365], x="date", y="sales", title='Sales record for Store 1 Item 1 in Year 2013')
fig.show()


y_2013[365:730]


fig = px.line(y_2013[365:730], x="date", y="sales", title='Sales record for Store 2 Item 1 in Year 2013')
fig.show()


y_2013[730:1095]


fig = px.line(y_2013[730:1095], x="date", y="sales", title='Sales record for Store 3 Item 1 in Year 2013')
fig.show()


monthvise= y_2013.groupby(y_2013[:365]['date'].dt.strftime('%B'))[['sales']].mean()
new_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August',
             'September', 'October', 'November', 'December']
monthvise = monthvise.reindex(new_order, axis=0)
monthvise
```

```
fig = go.Figure()

fig.add_trace(go.Bar(
    x=monthvise.index,
```

```
monthvise= y_2013.groupby(y_2013[365:730]['date'].dt.strftime('%B'))[['sales']].mean()
new_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August',
             'September', 'October', 'November', 'December']
monthvise = monthvise.reindex(new_order, axis=0)
monthvise
```

```
fig = go.Figure()

fig.add_trace(go.Bar(
    x=monthvise.index,
    y=monthvise['sales'],
    name='Stock Open Price',
    marker_color='crimson'
))
fig.update_layout(title='Average Sales for Store 2 Item 1 in year 2013')
```

```
sub.to_csv("Submission.csv",index=False)
```