```
#  Time Series Predicition using Deep Learning With  LSTM
#  ------------------------------------------------------

import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.model_selection import train_test_split
from keras.preprocessing.sequence import TimeseriesGenerator
import tensorflow as  tf

mpl.rcParams['figure.figsize'] = (10,8)
mpl.rcParams['axes.grid'] = False

df = pd.read_csv('/retail_sales_Time_series.csv')
df
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 52585,\n  \"fields\":
[\n    {\n      \"column\": \"Date\",\n      \"properties\": {\n
\"dtype\": \"object\",\n        \"num_unique_values\": 52585,\n
\"samples\": [\n          \"30/4/18 10:00\",\n          \"12/11/17
12:00\",\n          \"16/6/16 22:00\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Sales\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 29.957814641460093,\
n        \"min\": 32.79007417,\n        \"max\": 168.1072812,\n
\"num_unique_values\": 52585,\n        \"samples\": [\n
105.6581726,\n          96.49120044,\n          72.44353151\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52585 entries, 0 to 52584
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    52585 non-null  datetime64[ns]
 1   Sales   52585 non-null  float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 821.8 KB
```

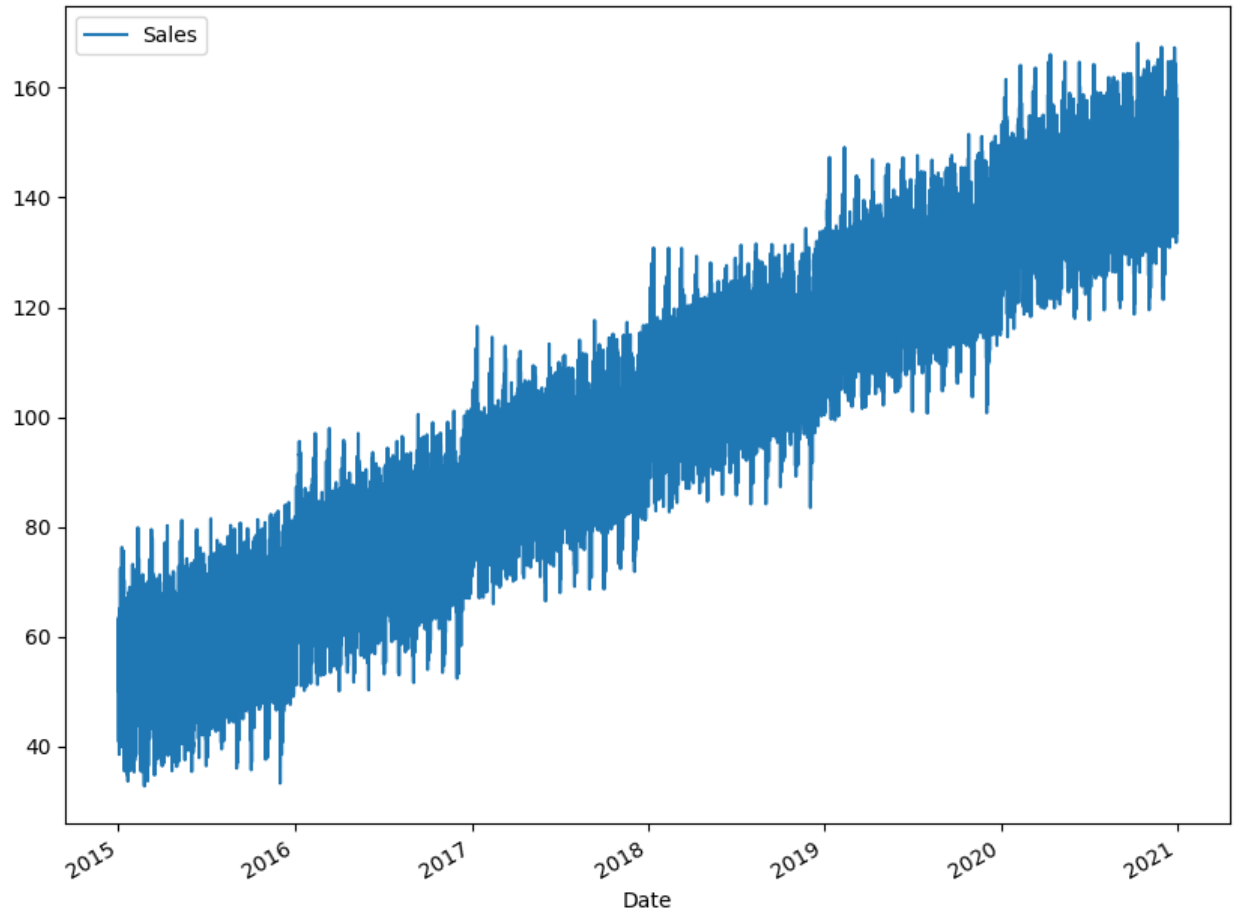
```
df['Date'] = pd.to_datetime(df['Date'], infer_datetime_format=True)
```

```
<ipython-input-8-45a55d54a14d>:1: UserWarning: The argument
'infer_datetime_format' is deprecated and will be removed in a future
version. A strict version of it is now the default, see
https://pandas.pydata.org/pdeps/0004-consistent-to-datetime-
```

```
parsing.html. You can safely remove this argument.
  df['Date'] = pd.to_datetime(df['Date'], infer_datetime_format=True)
<ipython-input-8-45a55d54a14d>:1: UserWarning: Could not infer format,
so each element will be parsed individually, falling back to
`dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
  df['Date'] = pd.to_datetime(df['Date'], infer_datetime_format=True)
```

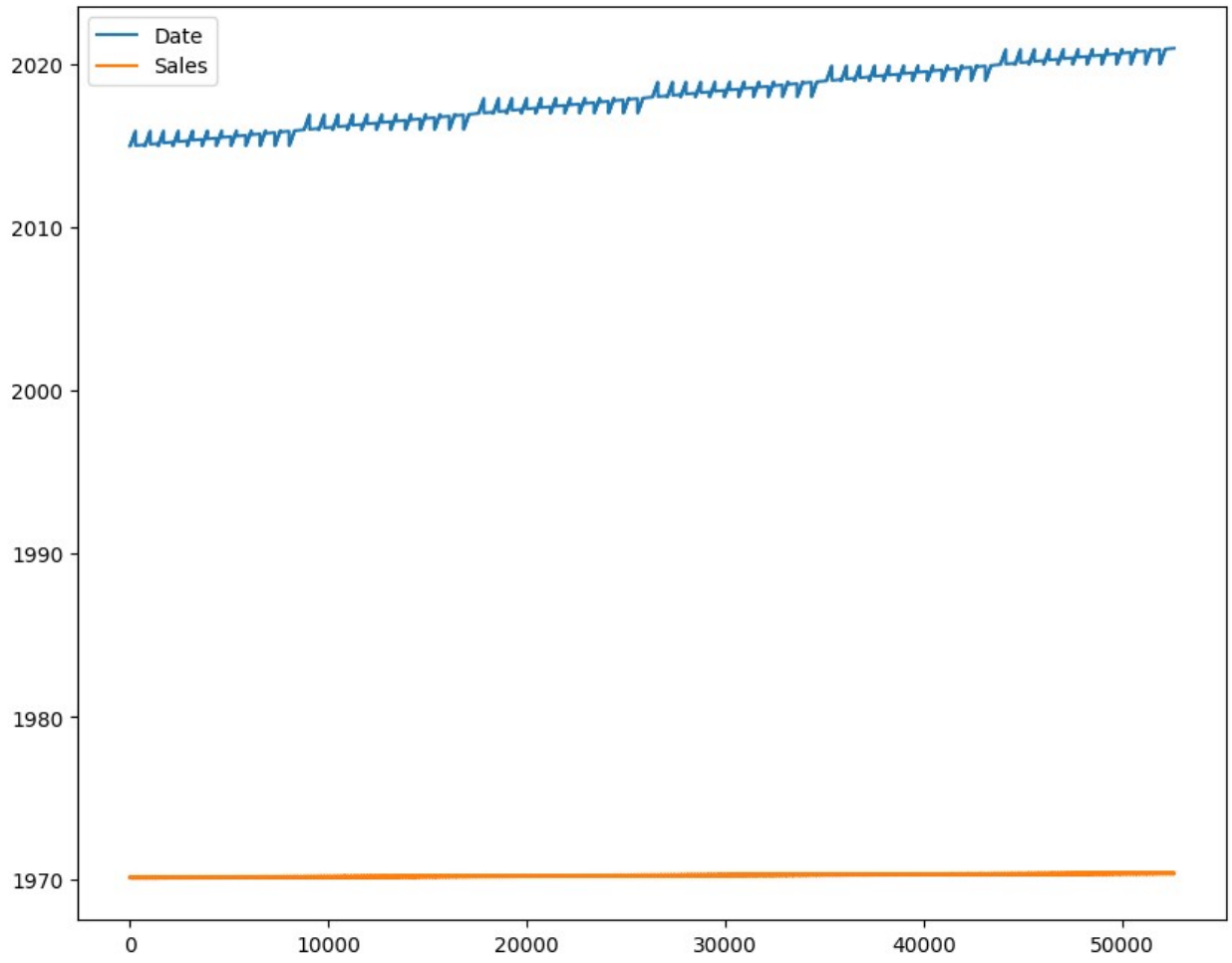
```
df.set_index('Date')[['Sales']].plot(subplots=True)
```

```
array([<Axes: xlabel='Date'>], dtype=object)
```




```
df.plot(figsize=(10,8))
```

```
<Axes: >
```




```
df_input=df[['Sales']]
df_input
```


```
{"summary":"{\n  \"name\": \"df_input\",\n  \"rows\": 52585,\n
\"fields\": [\n    {\n      \"column\": \"Sales\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
29.957814641460093,\n        \"min\": 32.79007417,\n        \"max\":
168.1072812,\n        \"num_unique_values\": 52585,\n
\"samples\": [\n          105.6581726,\n          96.49120044,\n
72.44353151\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_input"}
```


```
df_input.describe()
```


```
{"summary":"{\n  \"name\": \"df_input\",\n  \"rows\": 8,\n
\"fields\": [\n    {\n      \"column\": \"Sales\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
18559.801541552788,\n        \"min\": 29.957814641460093,\n
\"max\": 52585.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          99.9974882959477,\n          99.93064275,\n
```

```
52585.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
df_input.query('Sales>124')
```

```
{"summary":"{\n  \"name\": \"df_input\",\n  \"rows\": 13620,\n
\"fields\": [\n    {\n      \"column\": \"Sales\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9.308234762176067,\n        \"min\": 124.0014659,\n        \"max\":
168.1072812,\n        \"num_unique_values\": 13620,\n
\"samples\": [\n          128.09031,\n          132.9555999,\n
148.970802\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
scaler =MinMaxScaler()
data_scaled = scaler.fit_transform(df_input)
```

```
data_scaled98
```

```
array([[0.15710157],
       [0.1283763 ],
       [0.17118935],
       ...,
       [0.82542345],
       [0.85059557],
       [0.86783799]])
```

```
features=data_scaled
target=data_scaled[:,0]
```

```
TimeseriesGenerator(features,target,
length=2,sampling_rate=1,batch_size=1)[0]
```

```
(array([[[0.15710157],
         [0.1283763 ]]]),
 array([0.17118935]))
```

```
x_train, x_test, y_train, y_test  = train_test_split(features ,target,
test_size=0.20, random_state=125, shuffle=False)
```

```
x_train.shape
```

```
(42068, 1)
```

```
X_test.shape
```

```
(10517, 1)
```

```
win_length = 250
batch_size = 32
num_features = 1
train_generator = TimeseriesGenerator(x_train, y_train,
```

```
length=win_length, sampling_rate=1,batch_size=batch_size)
test_generator = TimeseriesGenerator(x_test , y_test,
length=win_length, sampling_rate=1,batch_size=batch_size)

train_generator[0]

(array([[[0.15710157],
         [0.1283763 ],
         [0.17118935],
         ...,
         [0.18276762],
         [0.22175255],
         [0.20120757]],

        [[0.1283763 ],
         [0.17118935],
         [0.17561098],
         ...,
         [0.22175255],
         [0.20120757],
         [0.14312201]],

        [[0.17118935],
         [0.17561098],
         [0.20468793],
         ...,
         [0.20120757],
         [0.14312201],
         [0.15166137]],

        ...,

        [[0.22315337],
         [0.24572904],
         [0.22854168],
         ...,
         [0.10166027],
         [0.06828254],
         [0.06927693]],

        [[0.24572904],
         [0.22854168],
         [0.2129003 ],
         ...,
         [0.06828254],
         [0.06927693],
         [0.0739013 ]],

        [[0.22854168],
         [0.2129003 ],
```

```
         [0.22877846],
         ...,
         [0.06927693],
         [0.0739013 ],
         [0.01741361]]]),
 array([0.14312201, 0.15166137, 0.1125318 , 0.10310585, 0.08499234,
        0.06207513, 0.04333615, 0.03926781, 0.03596528, 0.04262377,
        0.0642177 , 0.06894608, 0.10678641, 0.13736663, 0.10557498,
        0.13641166, 0.15753036, 0.17157231, 0.176193  , 0.19168563,
        0.18814053, 0.15758967, 0.17846883, 0.16403583, 0.12151806,
        0.11309371, 0.10166027, 0.06828254, 0.06927693, 0.0739013 ,
        0.01741361, 0.00389642]))
```

```
model =tf.keras.Sequential()
model.add(tf.keras.layers.LSTM(128, input_shape =
(win_length,num_features), return_sequences=True))
model.add(tf.keras.layers.LeakyReLU(alpha=0.5))
model.add(tf.keras.layers.LSTM(128,return_sequences=True))
model.add(tf.keras.layers.LeakyReLU(alpha=0.5))
model.add(tf.keras.layers.Dropout(0.3))
model.add(tf.keras.layers.LSTM(64,return_sequences=False))
model.add(tf.keras.layers.LeakyReLU(alpha=0.3))
model.add(tf.keras.layers.Dense(1))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 250, 128)          66560

 leaky_re_lu (LeakyReLU)     (None, 250, 128)          0

 lstm_1 (LSTM)               (None, 250, 128)          131584

 leaky_re_lu_1 (LeakyReLU)   (None, 250, 128)          0

 dropout (Dropout)           (None, 250, 128)          0

 lstm_2 (LSTM)               (None, 64)                49408

 leaky_re_lu_2 (LeakyReLU)   (None, 64)                0

 dense (Dense)               (None, 1)                 65

=================================================================
Total params: 247617 (967.25 KB)
Trainable params: 247617 (967.25 KB)
```

```
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
early_stopping =
tf.keras.callbacks.EarlyStopping(monitor='val_loss',patience=2,mode =
'min')

model.compile(loss= tf.losses.MeanSquaredError(),optimizer =
tf.optimizers.Adam(),metrics=[tf.metrics.MeanAbsoluteError()])

history = model.fit_generator(train_generator, epochs=50,
validation_data=test_generator, shuffle=False,
callbacks=[early_stopping])
```

```
Epoch 1/50

<ipython-input-48-85f216b38176>:5: UserWarning: `Model.fit_generator`
is deprecated and will be removed in a future version. Please use
`Model.fit`, which supports generators.
  history = model.fit_generator(train_generator, epochs=50,
validation_data=test_generator, shuffle=False,
callbacks=[early_stopping])

1307/1307 [==============================] - 1215s 925ms/step - loss:
8.7724e-04 - mean_absolute_error: 0.0223 - val_loss: 8.7840e-04 -
val_mean_absolute_error: 0.0232
Epoch 2/50
1307/1307 [==============================] - 1514s 1s/step - loss:
0.0013 - mean_absolute_error: 0.0184 - val_loss: 0.0022 -
val_mean_absolute_error: 0.0385
Epoch 3/50
1307/1307 [==============================] - 1343s 1s/step - loss:
3.9235e-04 - mean_absolute_error: 0.0154 - val_loss: 9.4191e-04 -
val_mean_absolute_error: 0.0246
```

```
model.evaluate_generator(test_generator, verbose=0)

<ipython-input-49-a68ec1088d46>:1: UserWarning:
`Model.evaluate_generator` is deprecated and will be removed in a
future version. Please use `Model.evaluate`, which supports
generators.
  model.evaluate_generator(test_generator, verbose=0)

[0.0009419145644642413, 0.024641329422593117]
```

```
predictions=model.predict_generator(test_generator)

<ipython-input-50-6ca49477046f>:1: UserWarning:
`Model.predict_generator` is deprecated and will be removed in a
future version. Please use `Model.predict`, which supports generators.
  predictions=model.predict_generator(test_generator)
```

```
predictions.shape[0]
```

```
10267
```

```
predictions
```

```
array([[0.77706367],
       [0.7703783 ],
       [0.75869006],
       ...,
       [0.7953903 ],
       [0.8085752 ],
       [0.82290584]], dtype=float32)
```

```
y_test
```

```
array([0.694982  , 0.67728269, 0.71390419, ..., 0.82542345,
0.85059557,
       0.86783799])
```

```
x_test
```

```
array([[0.694982  ],
       [0.67728269],
       [0.71390419],
       ...,
       [0.82542345],
       [0.85059557],
       [0.86783799]])
```

```
x_test[:,0:][win_length:]
```

```
array([[0.76973   ],
       [0.75606292],
       [0.76404814],
       ...,
       [0.82542345],
       [0.85059557],
       [0.86783799]])
```

```
df_pred=pd.concat([pd.DataFrame(predictions),pd.DataFrame(x_test[:,1:]
[win_length:])],axis=1)
```

```
df_pred
```


```
{"summary":"{\n  \"name\": \"df_pred\",\n  \"rows\": 10267,\n
\"fields\": [\n    {\n      \"column\": 0,\n      \"properties\": {\n
\"dtype\": \"float32\",\n        \"num_unique_values\": 10257,\n
\"samples\": [\n          0.6808833479881287,\n
0.7794239521026611,\n          0.8592292666435242\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df_pred"}
```

```
rev_trans=scaler.inverse_transform(df_pred)

rev_trans
```

```
array([[137.94017],
       [137.03552],
       [135.4539 ],
       ...,
       [140.42007],
       [142.20422],
       [144.1434 ]], dtype=float32)
```

```
df_final=df_input[predictions.shape[0]*-1:]

df_final.count()
```

```
Sales    10267
dtype: int64
```

```
df_final['Sales_pred']=rev_trans[:,0]
```

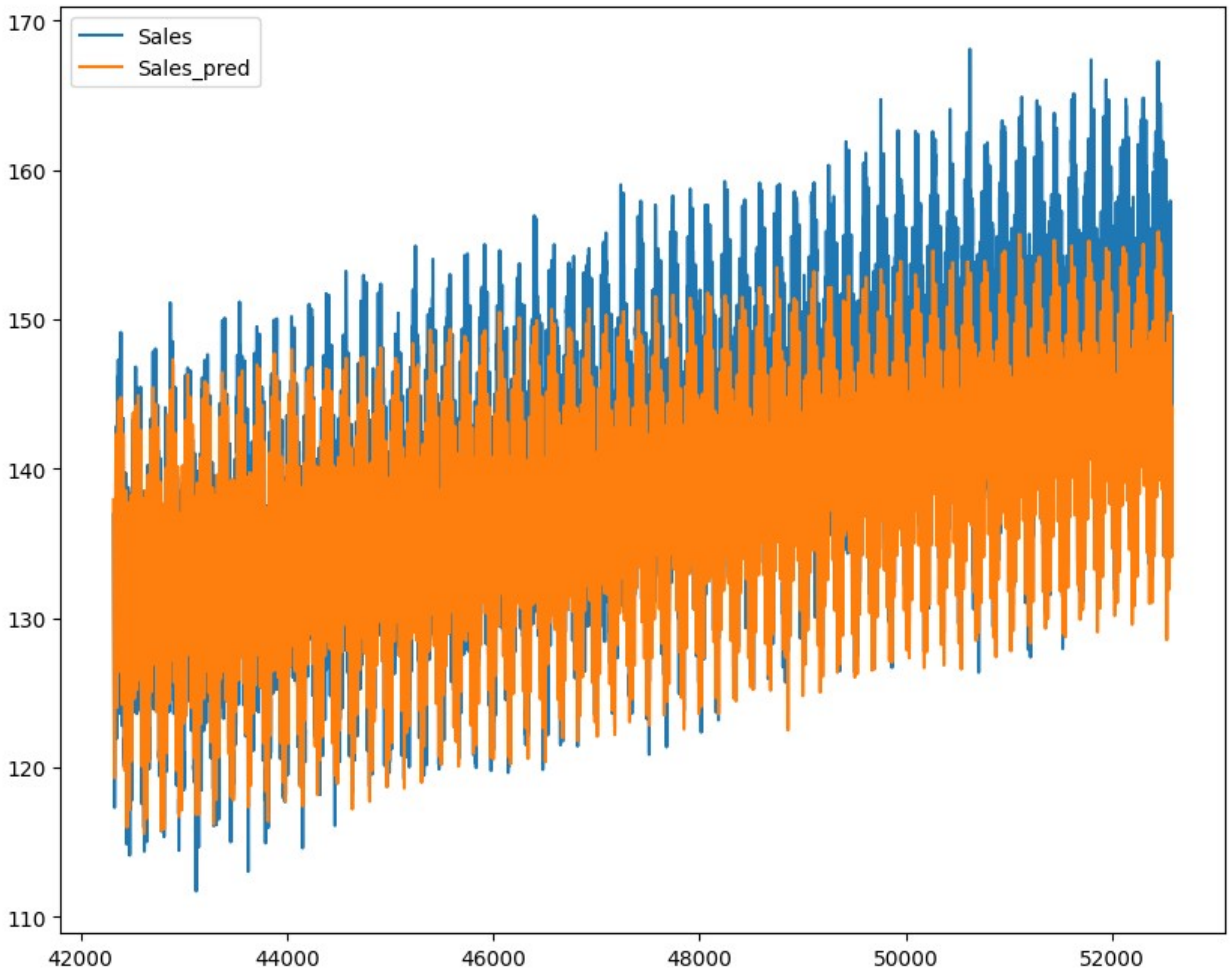
```
<ipython-input-66-1d01bceaffbb>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_final['Sales_pred']=rev_trans[:,0]
```

```
df_final
```


```
{"summary":"{\n  \"name\": \"df_final\",\n  \"rows\": 10267,\n
\"fields\": [\n    {\n      \"column\": \"Sales\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9.809966216969036,\n        \"min\": 111.7140685,\n        \"max\":
168.1072812,\n        \"num_unique_values\": 10267,\n
\"samples\": [\n          158.4069433,\n          155.9339592,\n
146.4885281\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Sales_pred\",\n      \"properties\": {\n        \"dtype\":
\"float32\",\n        \"num_unique_values\": 10247,\n
\"samples\": [\n          138.82313537597656,\n
123.15750885009766,\n          151.0559844970703\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df_final"}
```


```
df_final[['Sales','Sales_pred']].plot()
```

```
<Axes: >
```


Sales
Sales_pred
170
160
150
140
130
120
110
42000
44000
46000
48000
50000
52000