Accurate forecasting of demand can help the manufacturers to maintain appropriate stock which results in reduction in loss due to product not being sold and also reduces the opportunity cost (i.e. higher demand but less availability => opportunity lost)

Data fields

- **date** - Date of the sale data. There are no holiday effects or store closures.
- **store** - Store ID
- **item** - Item ID
- **sales** - Number of items sold at a particular store on a particular date.

In this project, the goal is to forecast 3-month sales for 50 different products in 10 different stores when given 5 years of store item sales data.

```
#import libraries
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn import metrics
import matplotlib.pyplot as plt
%matplotlib inline
from time import time

import warnings
warnings.filterwarnings("ignore")
```

```
# Read the dataset
dataset = pd.read_csv("item.csv")
dataset
```

	date	store	item	sales
0	2013-01-01	1	1	13
1	2013-01-02	1	1	11
2	2013-01-03	1	1	14
3	2013-01-04	1	1	13
4	2013-01-05	1	1	10
...	...	...	...	...
912995	2017-12-27	10	50	63
912996	2017-12-28	10	50	59
912997	2017-12-29	10	50	74
912998	2017-12-30	10	50	62
912999	2017-12-31	10	50	82

913000 rows × 4 columns

DATA CLEANING

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 913000 entries, 0 to 912999
Data columns (total 4 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   date    913000 non-null  object
 1   store   913000 non-null  int64
 2   item    913000 non-null  int64
 3   sales   913000 non-null  int64
dtypes: int64(3), object(1)
memory usage: 27.9+ MB
```

DATA HAS NO MISSING AND ZERO NULL VALUES FOR ALL COLUMNS SO NO NEED OF IMPUTE AND DROP THE DATA

```
#change the date column datatype object to datetime
from datetime import datetime, timedelta, date
dataset['date'] = pd.to_datetime(dataset['date'])
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 913000 entries, 0 to 912999
```

```
Data columns (total 4 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   date    913000 non-null  datetime64[ns]
 1   store   913000 non-null  int64
 2   item    913000 non-null  int64
 3   sales   913000 non-null  int64
dtypes: datetime64[ns](1), int64(3)
memory usage: 27.9 MB
```

Understanding Dataset

```
dataset.isnull().sum() # no null values and no duplicate rows
```

```
date     0
store    0
item     0
sales    0
dtype: int64
```

```
dataset['store'].unique()
#dataset.store.unique()
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
dataset.store.nunique()
```

```
10
```

```
dataset.groupby(["store"]).agg({"sales": ["count","sum", "mean", "median", "std", "min", "max"]})
```

	sales						
	count	sum	mean	median	std	min	max
store							
1	91300	4315603	47.268379	44.0	24.006252	1	155
2	91300	6120128	67.033165	62.0	33.595810	3	231
3	91300	5435144	59.530602	55.0	29.974102	3	196
4	91300	5012639	54.902946	51.0	27.733097	4	186
5	91300	3631016	39.770164	37.0	20.365757	2	130
6	91300	3627670	39.733516	37.0	20.310451	0	134
7	91300	3320009	36.363735	34.0	18.684825	1	122
8	91300	5856169	64.142048	60.0	32.231751	4	204
9	91300	5025976	55.049025	51.0	27.832186	4	195
10	91300	5360158	58.709288	54.0	29.554994	3	187

```
dataset.item.unique()
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50])
```

```
dataset.item.nunique()
```

```
50
```

```
dataset.groupby(["item"]).agg({"sales": ["count","sum", "mean", "median", "std", "min", "max"]})
```

	sales						
	count	sum	mean	median	std	min	max
item							
1	18260	401384	21.981599	21.0	8.468922	1	59
2	18260	1069564	58.574151	56.0	20.093015	9	150
3	18260	669087	36.642223	35.0	13.179441	7	104
4	18260	401907	22.010241	21.0	8.403898	0	66
5	18260	335230	18.358708	18.0	7.265167	1	50
6	18260	1068281	58.503888	56.0	20.174898	11	148
7	18260	1068777	58.531051	56.0	20.146002	11	141
8	18260	1405108	76.950055	74.0	26.130697	15	181
9	18260	938379	51.389869	49.5	17.790158	6	134
10	18260	1337133	73.227437	70.0	24.823725	14	175
11	18260	1271925	69.656353	67.0	23.744732	11	170
12	18260	1271534	69.634940	67.0	23.738663	12	170
13	18260	1539621	84.316594	81.0	28.311031	20	210
14	18260	1071531	58.681873	56.0	20.079860	12	152
15	18260	1607442	88.030778	85.0	29.522852	17	231
16	18260	468480	25.656079	25.0	9.603270	2	70
17	18260	602486	32.994852	32.0	11.967610	4	83
18	18260	1538876	84.275794	81.0	28.430621	18	208

Outliers

```
dataset.describe()
```

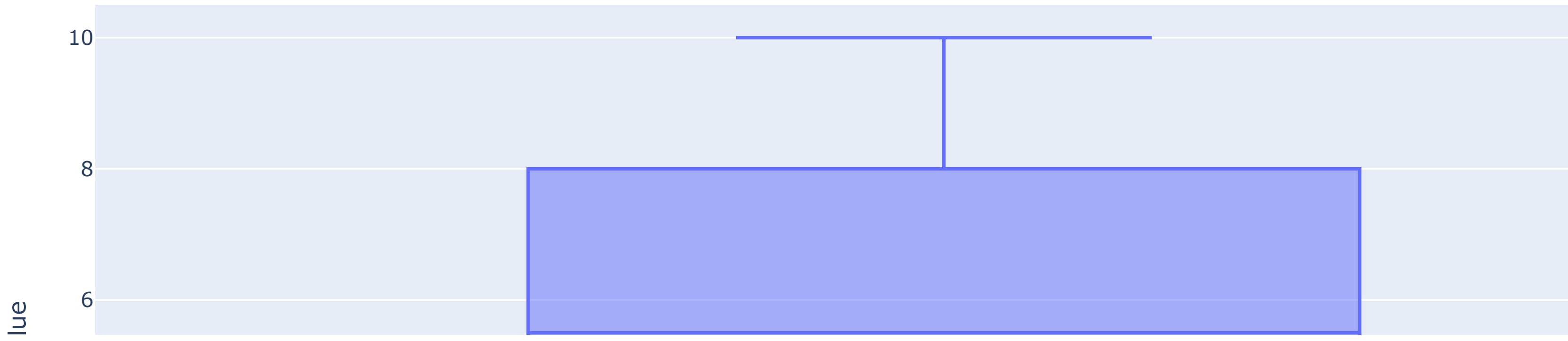
	store	item	sales
count	913000.000000	913000.000000	913000.000000
mean	5.500000	25.500000	52.250287
std	2.872283	14.430878	28.801144
min	1.000000	1.000000	0.000000
25%	3.000000	13.000000	30.000000
50%	5.500000	25.500000	47.000000
75%	8.000000	38.000000	70.000000
max	10.000000	50.000000	231.000000

```
iqr = dataset['store'].quantile(0.75) - dataset['store'].quantile(0.25)
upper_threshold = dataset['store'].quantile(0.75) + (1.5 * iqr)
lower_threshold = dataset['store'].quantile(0.25) - (1.5 * iqr)
upper_threshold, lower_threshold
```

```
(15.5, -4.5)
```

```
import plotly.express as px
from matplotlib.pyplot import figure

#sns.boxplot(dataset['store'])
fig = px.box(dataset["store"])
fig.show()
```

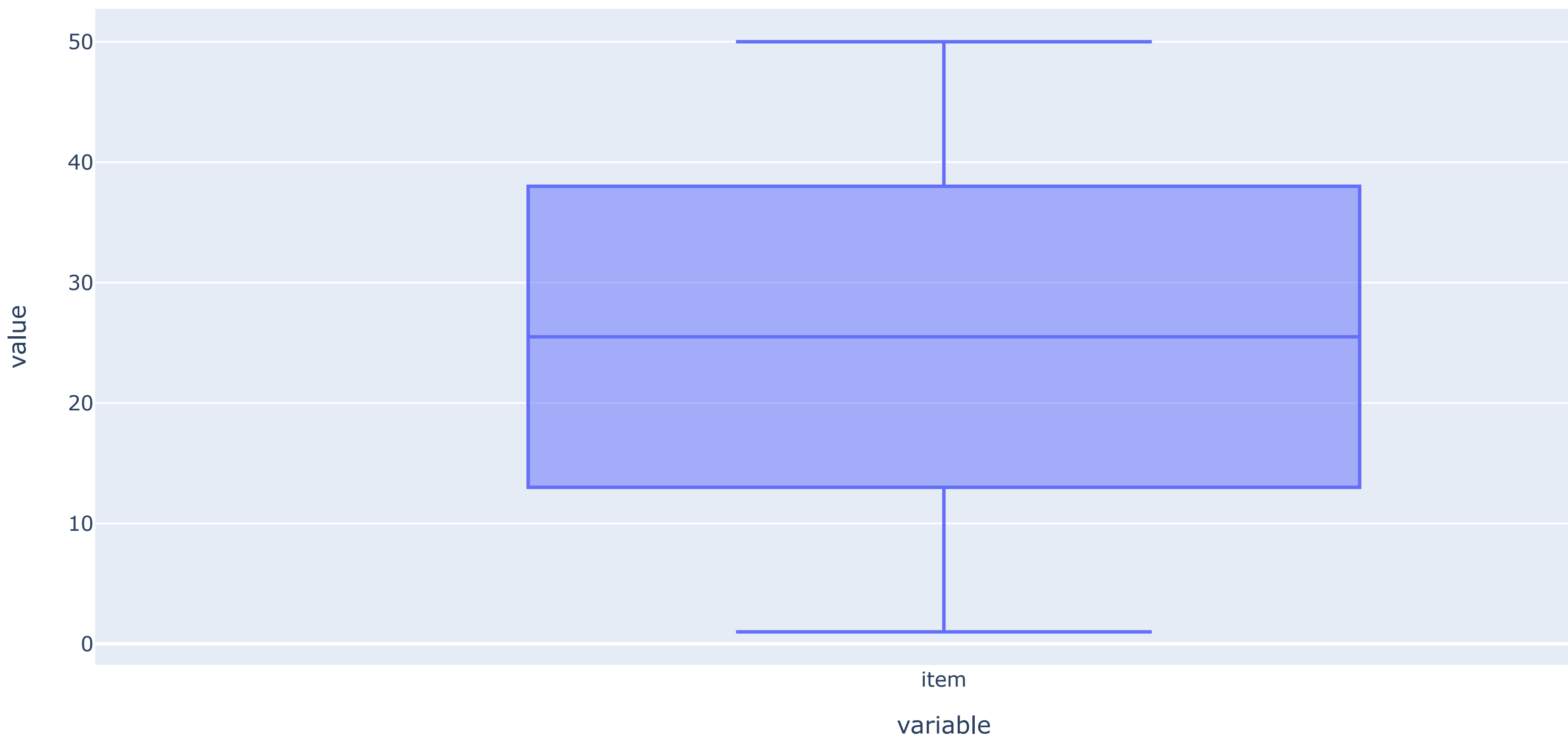


AS WE OBSERVED THE GRAPH THEIR IS NO OUTLIERS IN THE STORE

```
iqr = dataset['item'].quantile(0.75) - dataset['item'].quantile(0.25)
upper_threshold = dataset['item'].quantile(0.75) + (1.5 * iqr)
lower_threshold = dataset['item'].quantile(0.25) - (1.5 * iqr)
upper_threshold, lower_threshold
```

```
(75.5, -24.5)
```

```
#sns.boxplot(dataset['item'])
fig = px.box(dataset["item"])
fig.show()
```

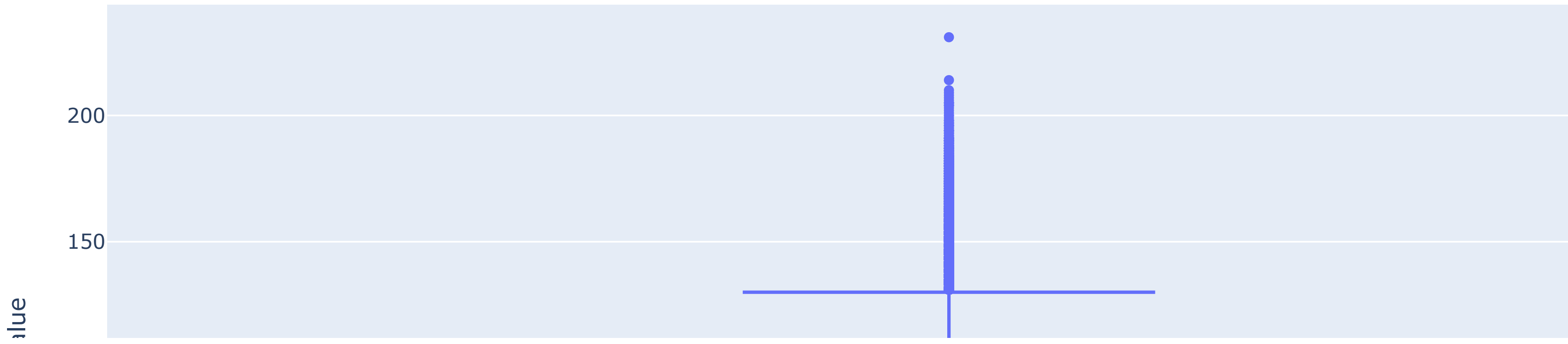


AS WE OBSERVED THE GRAPH THEIR IS NO OUTLIERS IN THE ITEM

```
iqr = dataset['sales'].quantile(0.75) - dataset['sales'].quantile(0.25)
upper_threshold = dataset['sales'].quantile(0.75) + (1.5 * iqr)
lower_threshold = dataset['sales'].quantile(0.25) - (1.5 * iqr)
upper_threshold, lower_threshold
```

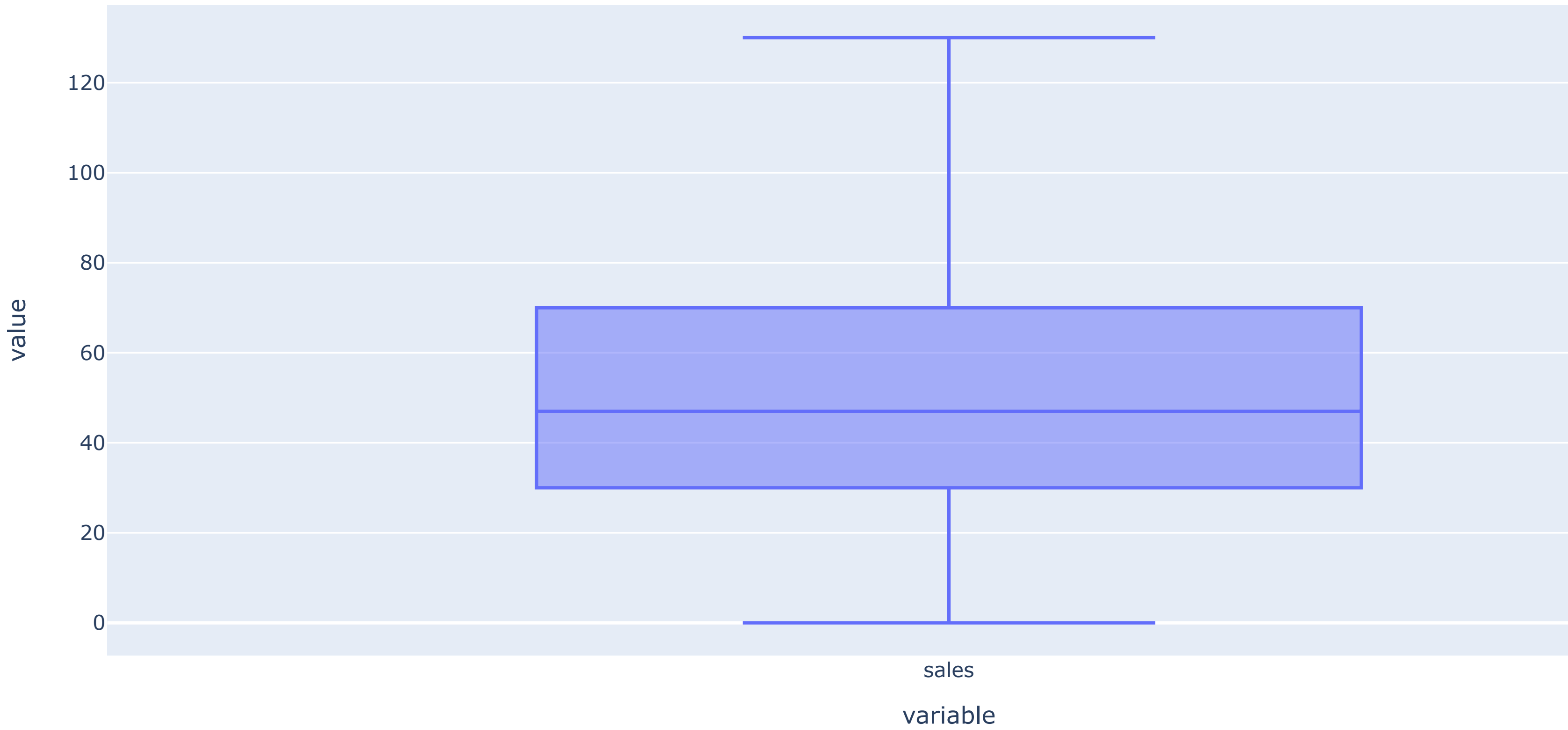
```
(130.0, -30.0)
```

```
fig = px.box(dataset["sales"])
fig.show()
```



AS WE OBSERVED THE GRAPH THEIR IS OUTLIERS IN THE SALES NEED TO CLIP

```
dataset['sales'] = dataset['sales'].clip(upper_threshold,lower_threshold)
dataset.sales
fig = px.box(dataset["sales"])
fig.show()
```



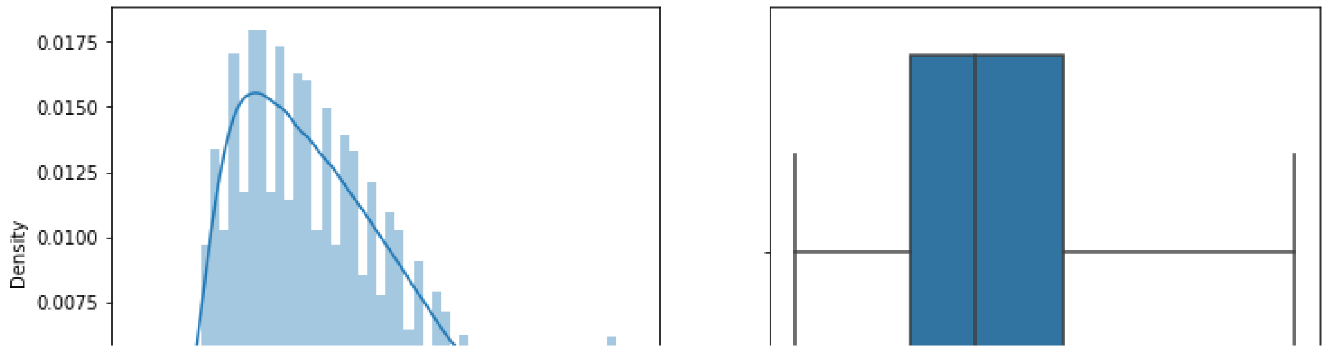
AFTER CLIPPING NO OUTLIERS FOUND

```
dataset.describe()
```

	store	item	sales
count	913000.000000	913000.000000	913000.000000
mean	5.500000	25.500000	52.067088
std	2.872283	14.430878	28.223040
min	1.000000	1.000000	0.000000
25%	3.000000	13.000000	30.000000
50%	5.500000	25.500000	47.000000
75%	8.000000	38.000000	70.000000
max	10.000000	50.000000	130.000000

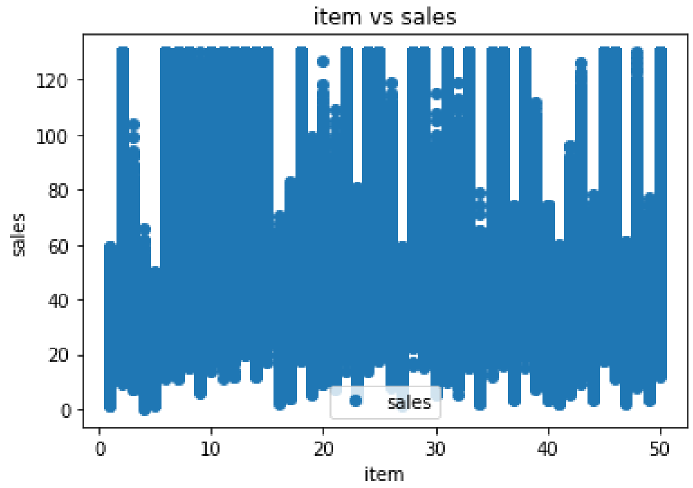
```
plt.subplots(figsize=(12, 5))
plt.subplot(1, 2, 1)
sns.distplot(dataset['sales'])

plt.subplot(1, 2, 2)
sns.boxplot(dataset['sales'])
plt.show()
```

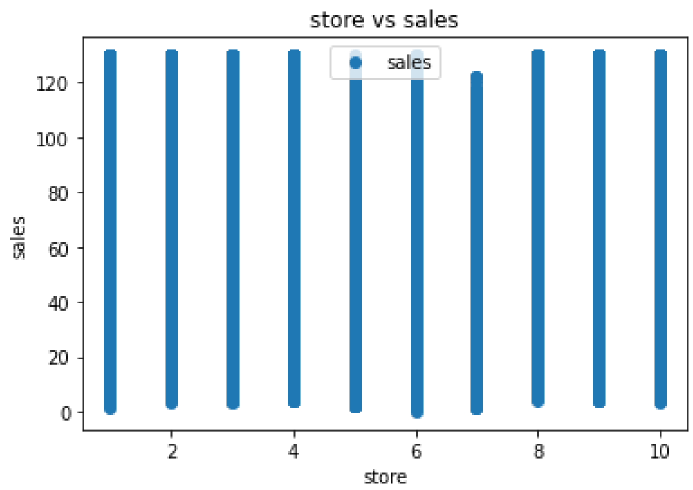


TASK JAR

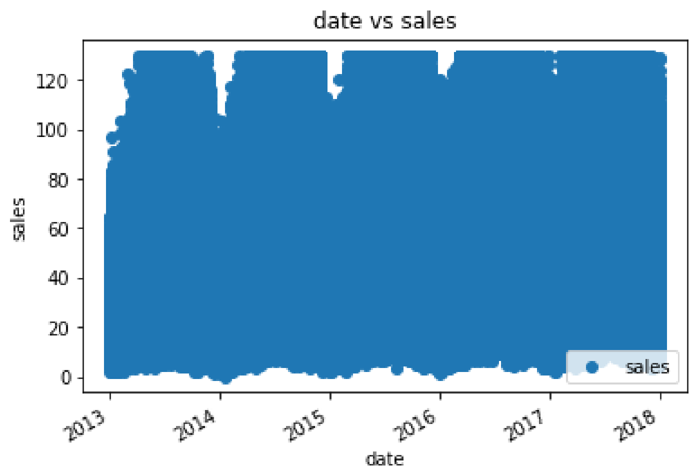
```
dataset.plot(x='item', y='sales', style='o')
plt.title('item vs sales')
plt.xlabel('item')
plt.ylabel('sales')
plt.show()
```




```
dataset.plot(x='store', y='sales', style='o')
plt.title('store vs sales')
plt.xlabel('store')
plt.ylabel('sales')
plt.show()
```




```
dataset.plot(x='date', y='sales', style='o')
plt.title('date vs sales')
plt.xlabel('date')
plt.ylabel('sales')
plt.show()
```

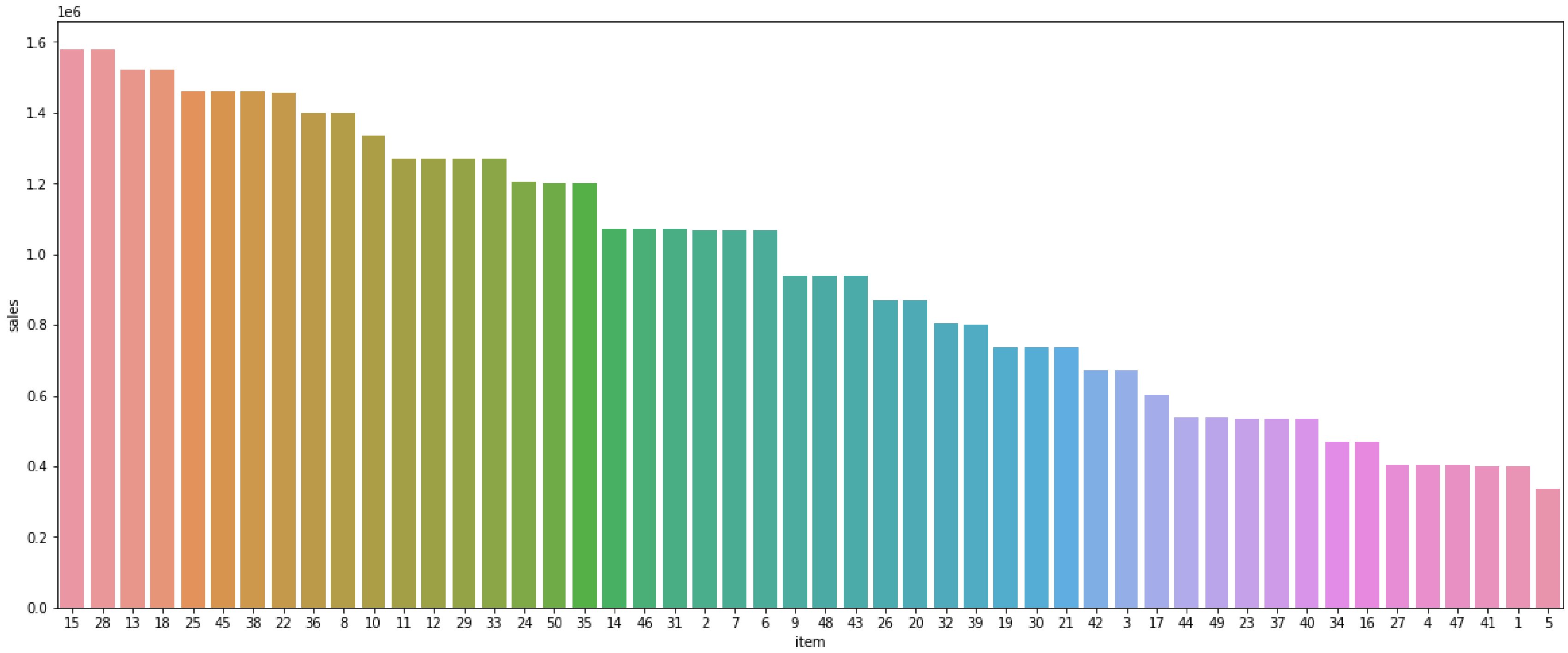



```
#Sales Data Per Item
sales_by_item = dataset.groupby('item')['sales'].sum().reset_index()
sales_by_item
```

	item	sales
0	1	401384.0
1	2	1069418.0
2	3	669087.0
3	4	401907.0
4	5	335230.0
5	6	1068156.0
6	7	1068743.0
7	8	1397852.0
8	9	938375.0
9	10	1333472.0
10	11	1269894.0
11	12	1269457.0
12	13	1521259.0
13	14	1071387.0
14	15	1580269.0
15	16	468480.0
16	17	602486.0
17	18	1520397.0
18	19	736892.0
19	20	867641.0
20	21	736190.0
21	22	1458608.0
22	23	534979.0
23	24	1205215.0
24	25	1461296.0
25	26	869981.0
26	27	402628.0
27	28	1577341.0
28	29	1269435.0
29	30	736554.0
30	31	1070640.0
31	32	803107.0
32	33	1268063.0
33	34	469935.0
34	35	1200760.0
35	36	1399638.0
36	37	534258.0
37	38	1458952.0
38	39	801311.0
39	40	534094.0
40	41	401759.0
41	42	669925.0
42	43	936635.0
43	44	536811.0
44	45	1459378.0
45	46	1070688.0
46	47	401781.0

```
fig, ax = plt.subplots(figsize=(20,8))
sns.barplot(sales_by_item.item, sales_by_item.sales, order=sales_by_item.sort_values('sales', ascending = False).item)
#ax.set(xlabel = "Item Id", ylabel = "Sum of Sales", title = "Total Sales Per Item")
```

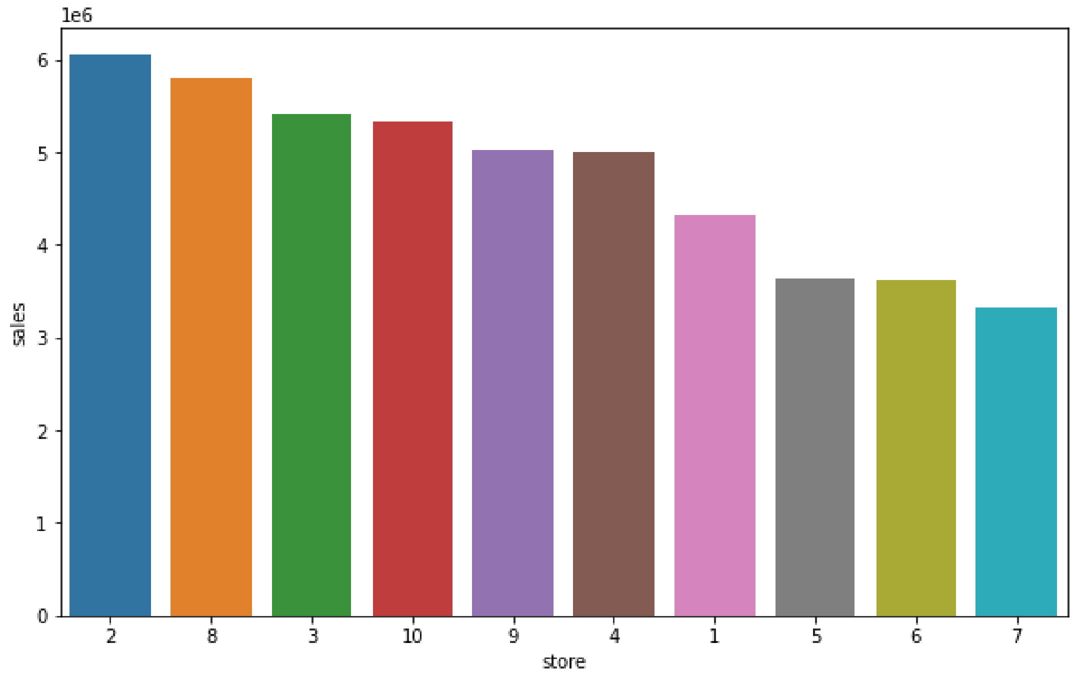
```
<AxesSubplot:xlabel='item', ylabel='sales'>
```




```
#Sales Data Per Store
sales_by_store = dataset.groupby('store')['sales'].sum().reset_index()


fig, ax = plt.subplots(figsize=(10,6))
sns.barplot(sales_by_store.store, sales_by_store.sales, order=sales_by_store.sort_values('sales',ascending = False).store)
#ax.set(xlabel = "Store Id", ylabel = "Sum of Sales", title = "Total Sales Per Store")
```

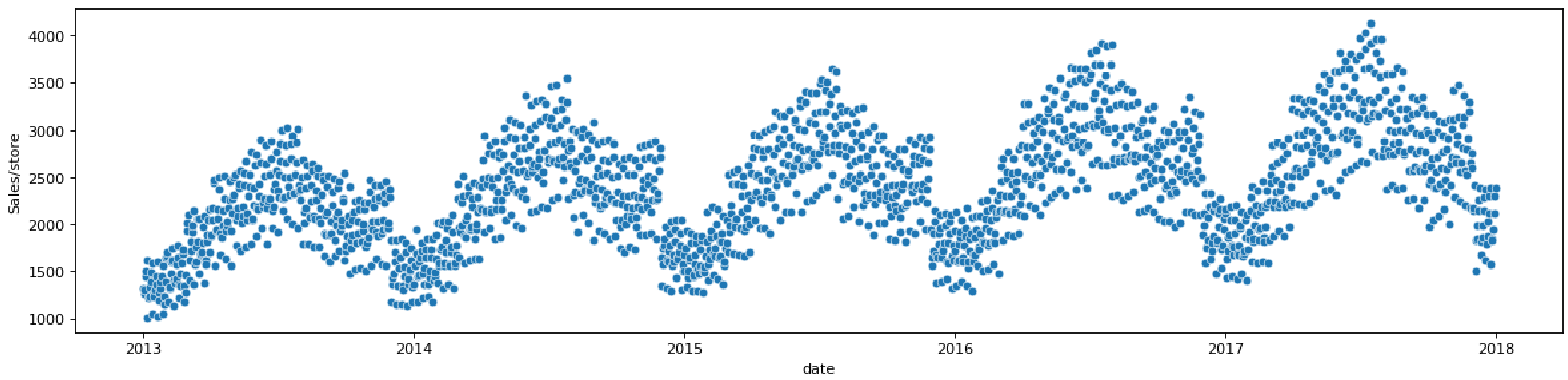
```
<AxesSubplot:xlabel='store', ylabel='sales'>
```




```
figure(figsize=(18, 4), dpi=80)
item_daily = dataset.groupby(["date","store"],as_index=False).agg({"sales":"sum"})

item_daily['date'] = pd.to_datetime(item_daily.date, format='%Y/%m/%d')
item_1 = item_daily[item_daily['store']==1]
ax_2 = sns.scatterplot(data=item_1,x='date',y='sales')
ax_2.set_ylabel("Sales/store")
```

```
Text(0, 0.5, 'Sales/store')
```




```
figure(figsize=(18, 4), dpi=80)
item_daily = dataset.groupby(["date","item"],as_index=False).agg({"sales":"sum"})

item_daily['date'] = pd.to_datetime(item_daily.date, format='%Y/%m/%d')
item_1 = item_daily[item_daily['item']==1]
ax_2 = sns.scatterplot(data=item_1,x='date',y='sales')
ax_2.set_ylabel("Sales/Item_1")
```

```
Text(0, 0.5, 'Sales/Item_1')
```

Feature Engineering

```
# Convert the date column to a datetime object
dataset['date'] = pd.to_datetime(dataset['date'])
```

```
# Create new columns for year, month, and day
dataset['year'] = dataset['date'].dt.year
dataset['month'] = dataset['date'].dt.month
dataset['day'] = dataset['date'].dt.day
```

```
from datetime import datetime
import calendar

def weekend_or_weekday(year,month,day):

    d = datetime(year,month,day)
    if d.weekday()>4:
        return 1
    else:
        return 0

dataset['weekend'] = dataset.apply(lambda x:weekend_or_weekday(x['year'], x['month'], x['day']), axis=1)
dataset.head()
```

	date	store	item	sales	year	month	day	weekend
0	2013-01-01	1	1	13.0	2013	1	1	0
1	2013-01-02	1	1	11.0	2013	1	2	0
2	2013-01-03	1	1	14.0	2013	1	3	0
3	2013-01-04	1	1	13.0	2013	1	4	0
4	2013-01-05	1	1	10.0	2013	1	5	1

```
def which_day(year, month, day):

    d = datetime(year,month,day)
    return d.weekday()

dataset['weekday'] = dataset.apply(lambda x: which_day(x['year'],x['month'],x['day']),axis=1)
dataset.head()
```

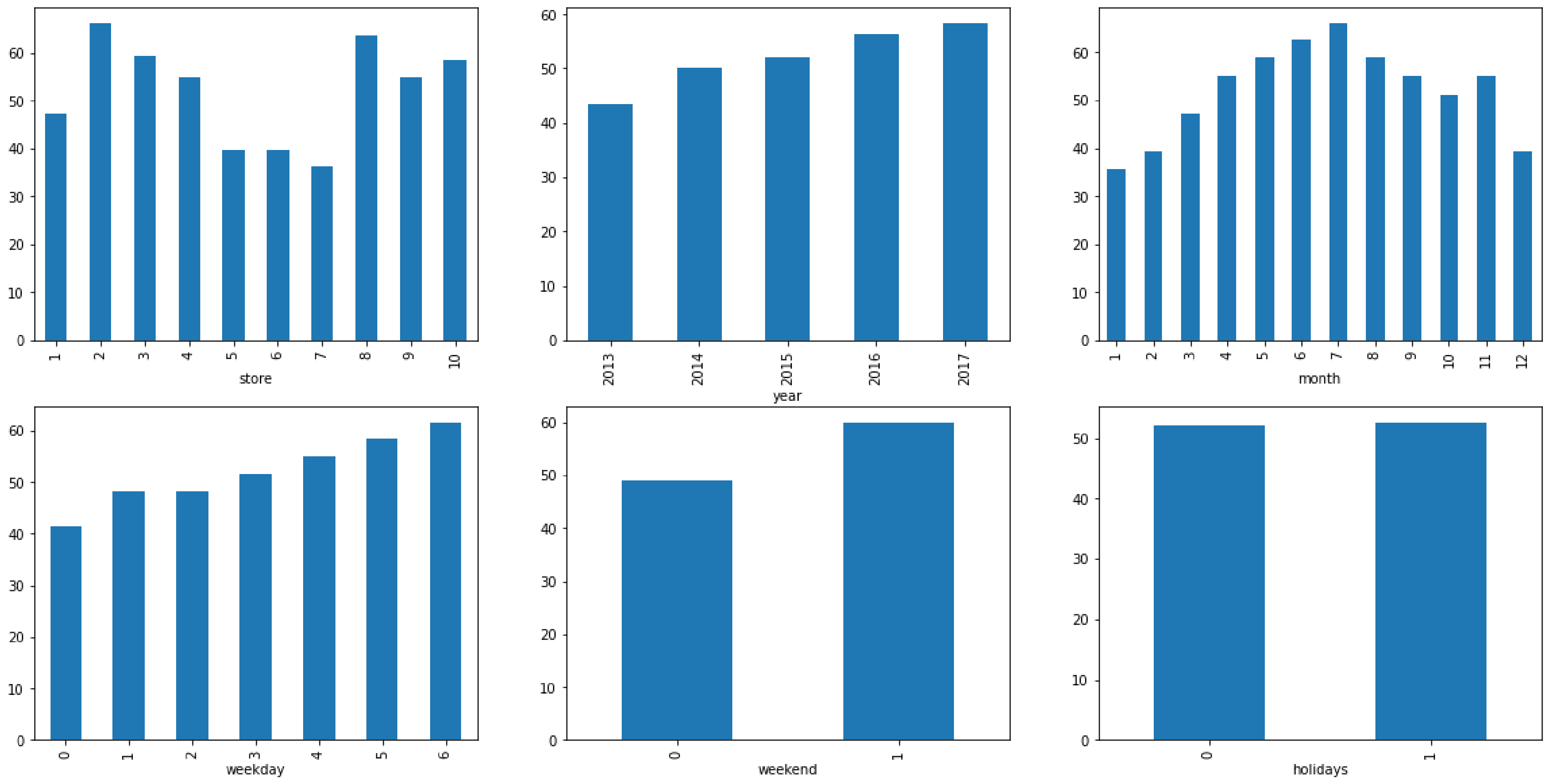
	date	store	item	sales	year	month	day	weekend	weekday
0	2013-01-01	1	1	13.0	2013	1	1	0	1
1	2013-01-02	1	1	11.0	2013	1	2	0	2
2	2013-01-03	1	1	14.0	2013	1	3	0	3
3	2013-01-04	1	1	13.0	2013	1	4	0	4
4	2013-01-05	1	1	10.0	2013	1	5	1	5

```
from datetime import date
import holidays
def is_holiday(x):
    india_holidays = holidays.country_holidays('IN')
    if india_holidays.get(x):
      return 1
    else:
      return 0
dataset['holidays'] = dataset['date'].apply(is_holiday)
dataset.head()
```

	date	store	item	sales	year	month	day	weekend	weekday	holidays
0	2013-01-01	1	1	13.0	2013	1	1	0	1	0
1	2013-01-02	1	1	11.0	2013	1	2	0	2	0

```
features = ['store', 'year', 'month',\
            'weekday', 'weekend','holidays' ]

plt.subplots(figsize=(20, 10))
for i, col in enumerate(features):
    plt.subplot(2, 3, i + 1)
    dataset.groupby(col).mean()['sales'].plot.bar()
plt.show()
```




```
item_i = dataset[dataset['item']==3]
item_i
```

	date	store	item	sales	year	month	day	weekend	weekday	holidays
36520	2013-01-01	1	3	15.0	2013	1	1	0	1	0
36521	2013-01-02	1	3	30.0	2013	1	2	0	2	0
36522	2013-01-03	1	3	14.0	2013	1	3	0	3	0
36523	2013-01-04	1	3	10.0	2013	1	4	0	4	0
36524	2013-01-05	1	3	23.0	2013	1	5	1	5	0
...	...	...	...	...	...	...	...	...	...	...
54775	2017-12-27	10	3	32.0	2017	12	27	0	2	0
54776	2017-12-28	10	3	33.0	2017	12	28	0	3	0
54777	2017-12-29	10	3	39.0	2017	12	29	0	4	0
54778	2017-12-30	10	3	34.0	2017	12	30	1	5	0
54779	2017-12-31	10	3	39.0	2017	12	31	1	6	0

18260 rows × 10 columns

```
k = item_i.groupby(['date','item'])
item_1 = k.agg(sum)
item_1=item_1.reset_index()
j=[]
for i in range(89, len(item_1)):
    b = item_1['date'][0+i] # 0 is the starting date and 0+i is the end date
    j.append(b)
item = item_1.head(1737) # doubt
item['end']=j    # doubt
date_list = dataset['date'].to_list()
d =[]
for i in range(1737):
     r = item.loc[i, 'end']
     a = date_list.index(r)
     c =item_1.loc[i:a,'sales'].sum()
```

```
    d.append(c)
item['total'] = d
item['date'] = pd.to_datetime(item['date'])
item['year'] = item['date'].dt.year
item['month'] = item['date'].dt.month
item['day'] = item['date'].dt.day
```

```
item
```

	date	item	store	sales	year	month	day	weekend	weekday	holidays	end	total
0	2013-01-01	3	55	172.0	2013	1	1	0	10	0	2013-03-31	21420.0
1	2013-01-02	3	55	213.0	2013	1	2	0	20	0	2013-04-01	21472.0
2	2013-01-03	3	55	193.0	2013	1	3	0	30	0	2013-04-02	21580.0
3	2013-01-04	3	55	218.0	2013	1	4	0	40	0	2013-04-03	21664.0
4	2013-01-05	3	55	217.0	2013	1	5	10	50	0	2013-04-04	21783.0
...	...	...	...	...	...	...	...	...	...	...	...	...
1732	2017-09-29	3	55	423.0	2017	9	29	0	40	0	2017-12-27	35038.0
1733	2017-09-30	3	55	482.0	2017	9	30	10	50	10	2017-12-28	34913.0
1734	2017-10-01	3	55	457.0	2017	10	1	10	60	0	2017-12-29	34763.0
1735	2017-10-02	3	55	332.0	2017	10	2	0	0	10	2017-12-30	34648.0
1736	2017-10-03	3	55	342.0	2017	10	3	0	10	0	2017-12-31	34718.0

1737 rows × 12 columns

Split the Data

```
x =  item.loc[:,['year','month','day']].values
y = item.loc[:,'total'].values
```

Train and Test the Data

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size =0.25)
```

Scale the Data

```
from sklearn.preprocessing import StandardScaler ## standrard scalig
scaler = StandardScaler() #initialise to a variable
scaler.fit(x_train) # we are finding the values of mean and sd from the td
x_train = scaler.transform(x_train) # fit (mean, sd) and then transform the training data
x_test= scaler.transform(x_test) # transform the test data
```

MODEL

Linear Regression

```
linear =LinearRegression()
linear.fit(x_train,y_train)
print('score for Linear Regression:',linear.score(x_test,y_test))
```

```
    score for Linear Regression: 0.44510958153315994
```

Decision Tree Regressor

```
from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score

for depth in [1,2,3,4,5,6,7,8,9,10,20,40,60]:
  dt = DecisionTreeRegressor(max_depth=depth)
  dt.fit(x_train,y_train)
  trainAccuracy = r2_score(y_train,dt.predict(x_train))
  dt = DecisionTreeRegressor(max_depth = depth)
  valAccuracy = cross_val_score(dt, x_train, y_train, cv=10, scoring = make_scorer(r2_score))
  print("Depth:",depth,'Train R2:',trainAccuracy,'Val Score:',np.mean(valAccuracy))
dt = DecisionTreeRegressor(max_depth = int(input('max depth value')))
```

```
dt.fit(x_train,y_train)
#print('score for Decision Treeregressor:',dt.score(x_test,y_test))
```

```
Depth: 1 Train R2: 0.35087341198894495 Val Score: 0.33381460440911626
Depth: 2 Train R2: 0.5655852496160495 Val Score: 0.5518352189862905
Depth: 3 Train R2: 0.7572635091542361 Val Score: 0.7486419366195585
Depth: 4 Train R2: 0.8696316259916675 Val Score: 0.8639621714011867
Depth: 5 Train R2: 0.9352874962083143 Val Score: 0.9295265858645527
Depth: 6 Train R2: 0.9656008040289864 Val Score: 0.9605280485872081
Depth: 7 Train R2: 0.985761055284366 Val Score: 0.9819599024081139
Depth: 8 Train R2: 0.9939897995249319 Val Score: 0.9913427511074309
Depth: 9 Train R2: 0.9975325212898226 Val Score: 0.9953882233716881
Depth: 10 Train R2: 0.999293975286992 Val Score: 0.9974006217360276
Depth: 20 Train R2: 1.0 Val Score: 0.9982747652291255
Depth: 40 Train R2: 1.0 Val Score: 0.9982747652291255
Depth: 60 Train R2: 1.0 Val Score: 0.9982747652291255
max depth value7
▾ DecisionTreeRegressor
DecisionTreeRegressor(max_depth=7)
```

KNeighborsRegressor

```
#from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score
for i in [1,2,3,4,5,6,7,8,9,10,20,50]:
  knn = KNeighborsRegressor(i)
  knn.fit(x_train,y_train)
  print('k value:',i ,'train score:',knn.score(x_train,y_train),'cv score:',np.mean(cross_val_score(knn, x_train, y_train, cv=10, scoring = make_scc
knn =KNeighborsRegressor(int(input('enter k values:')))
knn.fit(x_train,y_train)
#print('score for knn regression :',knn.score(x_test,y_test))
```

```
k value: 1 train score: 1.0 cv score: 0.9944978503833447
k value: 2 train score: 0.9993762103272856 cv score: 0.9906783263898694
k value: 3 train score: 0.9976417334112342 cv score: 0.9850208260723383
k value: 4 train score: 0.9944996094928108 cv score: 0.9806985541092474
k value: 5 train score: 0.9900037407566529 cv score: 0.9769560347394963
k value: 6 train score: 0.986057912803713 cv score: 0.9754990989573373
k value: 7 train score: 0.9840212674773232 cv score: 0.9747118439229763
k value: 8 train score: 0.9832716856582698 cv score: 0.9728998563932496
k value: 9 train score: 0.9817098441466363 cv score: 0.9701353052315124
k value: 10 train score: 0.9796293804897782 cv score: 0.9684069130034093
k value: 20 train score: 0.9609551607853506 cv score: 0.9496938223879425
k value: 50 train score: 0.9317028171722405 cv score: 0.9179396031781899
enter k values:9
▾ KNeighborsRegressor
KNeighborsRegressor(n_neighbors=9)
```

```
!pip install -U scikit-learn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.8/dist-packages (1.2.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-learn) (3.1.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.8/dist-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.8/dist-packages (from scikit-learn) (1.22.4)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.8/dist-packages (from scikit-learn) (1.10.1)
```

XGBRegressor

```
from xgboost import XGBRegressor
import xgboost as xgb

for lr in [0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.1,0.11,0.12,0.13,0.14,0.15,0.2,0.5,0.7,1]:
  model = xgb.XGBRegressor(learning_rate = lr,n_estimators =100,verbosity =0)#initialise the model
  model.fit(x_train,y_train)
  model.score(x_test,y_test)
  print('Learning rate:',lr,"Train score",model.score(x_train,y_train),'Cross-Val score:',np.mean(cross_val_score(model,x_train,y_train,cv=10)))
model = xgb.XGBRegressor(learning_rate = float(input('LR value')),n_estimators =100)
model.fit(x_train,y_train)

print('score for the XGBRegressor:',model.score(x_test,y_test))
```

```
Learning rate: 0.01 Train score -3.543000307732897 Cross-Val score: -3.661468404912412
Learning rate: 0.02 Train score 0.3707805574018431 Cross-Val score: 0.3513795595029888
Learning rate: 0.03 Train score 0.909057023905384 Cross-Val score: 0.9042564981695304
Learning rate: 0.04 Train score 0.985658851935628 Cross-Val score: 0.9835801055361204
Learning rate: 0.05 Train score 0.9969948251879686 Cross-Val score: 0.9955654530949556
Learning rate: 0.06 Train score 0.9990276828917134 Cross-Val score: 0.9979306746090411
Learning rate: 0.07 Train score 0.9995081366041055 Cross-Val score: 0.9985166531733516
Learning rate: 0.08 Train score 0.9996570369472374 Cross-Val score: 0.9988872717525119
Learning rate: 0.09 Train score 0.9997110494646698 Cross-Val score: 0.9990153891017117
Learning rate: 0.1 Train score 0.9998008554352767 Cross-Val score: 0.9991960272757888
Learning rate: 0.11 Train score 0.9998351944062677 Cross-Val score: 0.999294055610536
Learning rate: 0.12 Train score 0.9998683883238616 Cross-Val score: 0.9993689517412582
Learning rate: 0.13 Train score 0.9998724883198012 Cross-Val score: 0.9993575369433663
```

```
Learning rate: 0.14 Train score 0.9998845218067909 Cross-Val score: 0.9993956949207004
Learning rate: 0.15 Train score 0.9998961275652046 Cross-Val score: 0.9994560313721266
Learning rate: 0.2 Train score 0.9999169030164518 Cross-Val score: 0.9994547278384832
Learning rate: 0.5 Train score 0.9999770263823476 Cross-Val score: 0.999320163883489
Learning rate: 0.7 Train score 0.9999858021737482 Cross-Val score: 0.9989570921819642
Learning rate: 1 Train score 0.9999895539418875 Cross-Val score: 0.998211953105257
LR value1
score for the XGBRegressor: 0.9988394789713089
```

Accuracy

```
print('Score for the Linear Regressor       :',linear.score(x_test,y_test))
print('Score for the Decision TreeRegressor :',dt.score(x_test,y_test))
print('Score for the KNN Regressor          :',knn.score(x_test,y_test))
print('Score for the XGBRegressor           :',model.score(x_test,y_test))
#print('score for the Random Forest:',RF.score(x_test,y_test))
```

```
Score for the Linear Regressor       : 0.44510958153315994
Score for the Decision TreeRegressor : 0.9845198675189734
Score for the KNN Regressor          : 0.9738434304408886
Score for the XGBRegressor           : 0.9988394789713089
```

```
linear_pred = linear.predict(x_test)
dt_pred= dt.predict(x_test)
knn_pred=knn.predict(x_test)
xgb_pred=model.predict(x_test)
```

```
pd.DataFrame({"Actual":y_test, "linear_pred":linear_pred,"dt_pred":dt_pred, "knn_pred":knn_pred,"xgb_pred":xgb_pred })
```

	Actual	linear_pred	dt_pred	knn_pred	xgb_pred
0	32281.0	32473.388222	31799.000000	32516.111111	32184.464844
1	39803.0	34761.651582	39272.289855	38356.222222	39735.714844
2	36955.0	32084.808817	37521.315068	36426.444444	37109.875000
3	31006.0	38582.314248	32230.285714	33459.444444	31131.289062
4	36103.0	37602.916900	34777.227273	37270.111111	36433.335938
...	...	...	...	...	...
430	35935.0	38167.832525	35476.500000	36273.000000	36162.148438
431	33059.0	28562.398395	32697.173913	32769.777778	32895.082031
432	22106.0	25901.163841	22196.333333	22716.333333	22181.900391
433	42849.0	36642.115766	43175.279412	42095.000000	42877.011719
434	21580.0	30734.432277	21678.200000	24058.111111	21313.500000

435 rows × 5 columns