Introduction

The M/S Kambar Fireworks is a wholesale / retail sole proprietorship concern incorporated in Kerala, India. They do have multiple sales channels, viz over the counter (OTC), wholesale agents, retail outlets and affiliates etc to name a few. As part of this project, by liasoning with the IT department of the firm, over the counter (OTC) data from April 2015 through March 2018 (three consecutive years) were procured as comma separated vector (CSV) files. This report is a summary of a list of activities undertaken to do the sales forecasting with the procured data. The nature of OTC business is that it is a seasonal one. The activities are centered around the key festivals which are in vogue across the India.

Data Set

1. Product
   productId : Unique Id corresponding to a product
   productName: Product Name
   productCategory: Product Category
   rate: rate of the product which may change over the peroid

2. InvoiceMaster
   billId: Unique id corresponding to a bill
   date: date the bill generated
   total: total amount of the bill
   retailerId: Unique Id corresponding to a retailer

3. InvoiceFacts
   billId: id corresponding to a bill
   skuId: id corresponding to a product
   quantity: quantity of items purchased
   rate: rate of the item which may change over the peroid
   amount: product of rate and quantity

In [1]:

```
# import libraries

import pandas as pd
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV, Randomiz
from numpy import mean, std
from pprint import pprint
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error

from sklearn.tree import DecisionTreeRegressor
```

```python
from sklearn import metrics, ensemble
from sklearn.linear_model import LinearRegression
```

In [2]:
```python
# load the dataset
product = pd.read_csv("Product.csv")
invoiceMaster = pd.read_csv("InvoiceMaster.csv")
invoiceFacts = pd.read_csv("InvoiceFacts.csv")
```

Exploratory Data Analysis And Data Preparation Using Pandas

In [3]:
```python
# Print the information about the product dataframe
product.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 79 entries, 0 to 78
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   productId        79 non-null     int64 
 1   productName      79 non-null     object
 2   productCategory  79 non-null     object
 3   rate             79 non-null     int64 
dtypes: int64(2), object(2)
memory usage: 2.6+ KB
```

In [4]:
```python
# Print first few records of product data frame
product.head()
```

Out[4]:

	productId	productName	productCategory	rate
0	1	Ground Chakra Medium 7.5 inch 10 Per Box	Chakram	70
1	2	Ground Chakra Big 10.5 inch 10 Per Box	Chakram	81
2	3	Ground Chakra Asoka 21 inch 10 Per Box	Chakram	152
3	4	Ground Chakra Delux 31 inch 10 Per Box	Chakram	288
4	5	Twinkling Star 1.5 inch 10 Per Box	Twinkling Star	75

In [5]:
```python
# Print the information about the invoiceMaster dataframe
invoiceMaster.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29806 entries, 0 to 29805
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   billId       29806 non-null  int64 
 1   date         29806 non-null  object
 2   retailerId   29806 non-null  int64 
 3   total        29806 non-null  int64 
dtypes: int64(3), object(1)
memory usage: 931.6+ KB
```

In [6]:
```python
# Print first few records of invoiceMaster data frame
invoiceMaster.head()
```

Out[6]:

	billId	date	retailerId	total
0	29029	01/04/2015	4	3140
1	29030	01/04/2015	4	1055
2	29031	01/04/2015	4	1130
3	29032	01/04/2015	4	400
4	29033	01/04/2015	4	480

In [7]:
```python
# Print the information about the invoiceFacts dataframe
invoiceFacts.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 122346 entries, 0 to 122345
Data columns (total 6 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   billId      122346 non-null  int64
 1   sequenceNo  122346 non-null  int64
 2   skuId       122346 non-null  int64
 3   quantity    122346 non-null  int64
 4   rate        122346 non-null  int64
 5   amount      122346 non-null  int64
dtypes: int64(6)
memory usage: 5.6 MB
```

In [8]:
```python
# Print first few records of invoiceFacts data frame
invoiceFacts.head()
```

Out[8]:

	billId	sequenceNo	skuId	quantity	rate	amount
0	29029	1	70	2	670	1340
1	29029	2	60	1	405	405
2	29029	3	9	2	175	350
3	29029	4	34	3	60	180
4	29029	5	62	5	85	425

In [9]:
```python
product_train = product.copy()
# rename the productId to skuId for Product
product_train = product_train.rename(columns={'productId': 'skuId'})
# drop rate column as this change year
product_train = product_train.drop(columns=['rate'])
product_train.head()
```

Out[9]:

	skuId	productName	productCategory
0	1	Ground Chakra Medium 7.5 inch 10 Per Box	Chakram
1	2	Ground Chakra Big 10.5 inch 10 Per Box	Chakram
2	3	Ground Chakra Asoka 21 inch 10 Per Box	Chakram
3	4	Ground Chakra Delux 31 inch 10 Per Box	Chakram
4	5	Twinkling Star 1.5 inch 10 Per Box	Twinkling Star

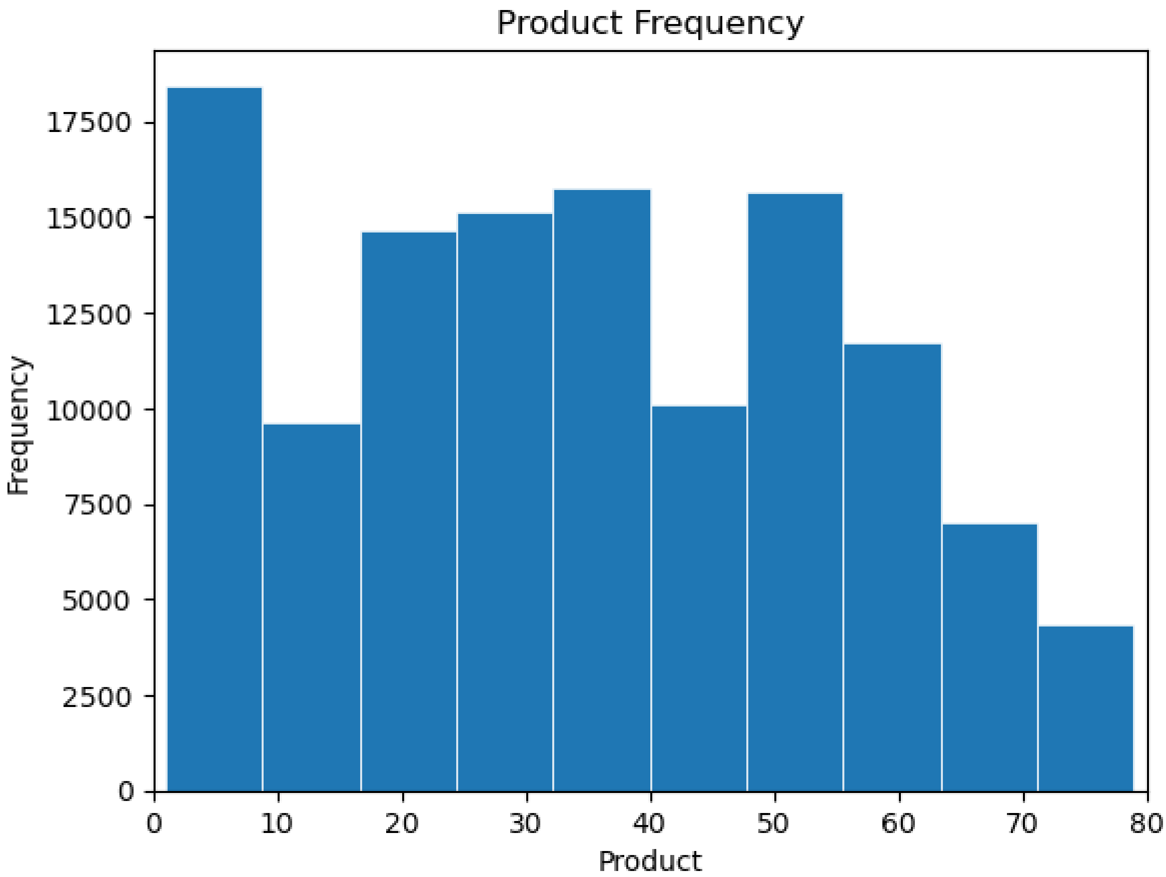
In [10]:
```python
# merge invoiceMaster and product based on skuId
train = invoiceFacts.merge(product_train, on = 'skuId', how = 'left')
train.head()
```

Out[10]:

	billId	sequenceNo	skuId	quantity	rate	amount	productName	productCategory
0	29029	1	70	2	670	1340	Aerials Green Boom 3 per Box	Aerials
1	29029	2	60	1	405	405	Knife 100 Shots	Shots
2	29029	3	9	2	175	350	Coronation Candle 10 Per Box	Pencils & Stones
3	29029	4	34	3	60	180	Bijili Crackers Red 50 Per Box	Crackers
4	29029	5	62	5	85	425	Roll Caps	Roll Caps

In [11]:

```
# plot frequency table of product
x = invoiceFacts.skuId
# plot:
fig, ax = plt.subplots()

ax.hist(x, bins=10, linewidth=0.5, edgecolor="white")

ax.set(xlim=(0, 80))
ax.set_xlabel('Product')
ax.set_ylabel('Frequency')
ax.set_title("Product Frequency")
plt.show()
```



In [12]:

```
# merge invoiceMaster and train based on billId
train = train.merge(invoiceMaster, on = 'billId', how = 'left')
train.head()
```

Out[12]:

	billId	sequenceNo	skuId	quantity	rate	amount	productName	productCategory	
0	29029	1	70	2	670	1340	Aerials Green Boom 3 per Box	Aerials	01/(
1	29029	2	60	1	405	405	Knife 100 Shots	Shots	01/(
2	29029	3	9	2	175	350	Coronation Candle 10 Per Box	Pencils & Stones	01/(
3	29029	4	34	3	60	180	Bijili Crackers Red 50 Per Box	Crackers	01/(
4	29029	5	62	5	85	425	Roll Caps	Roll Caps	01/(

In [13]:

```
# convert date string to datetime and
# add year and month columns to train data frame
train['date'] = pd.to_datetime(train["date"], format='%d/%m/%Y')
train['year'] = pd.DatetimeIndex(train['date']).year
train['month'] = pd.DatetimeIndex(train['date']).month
train.head()
```

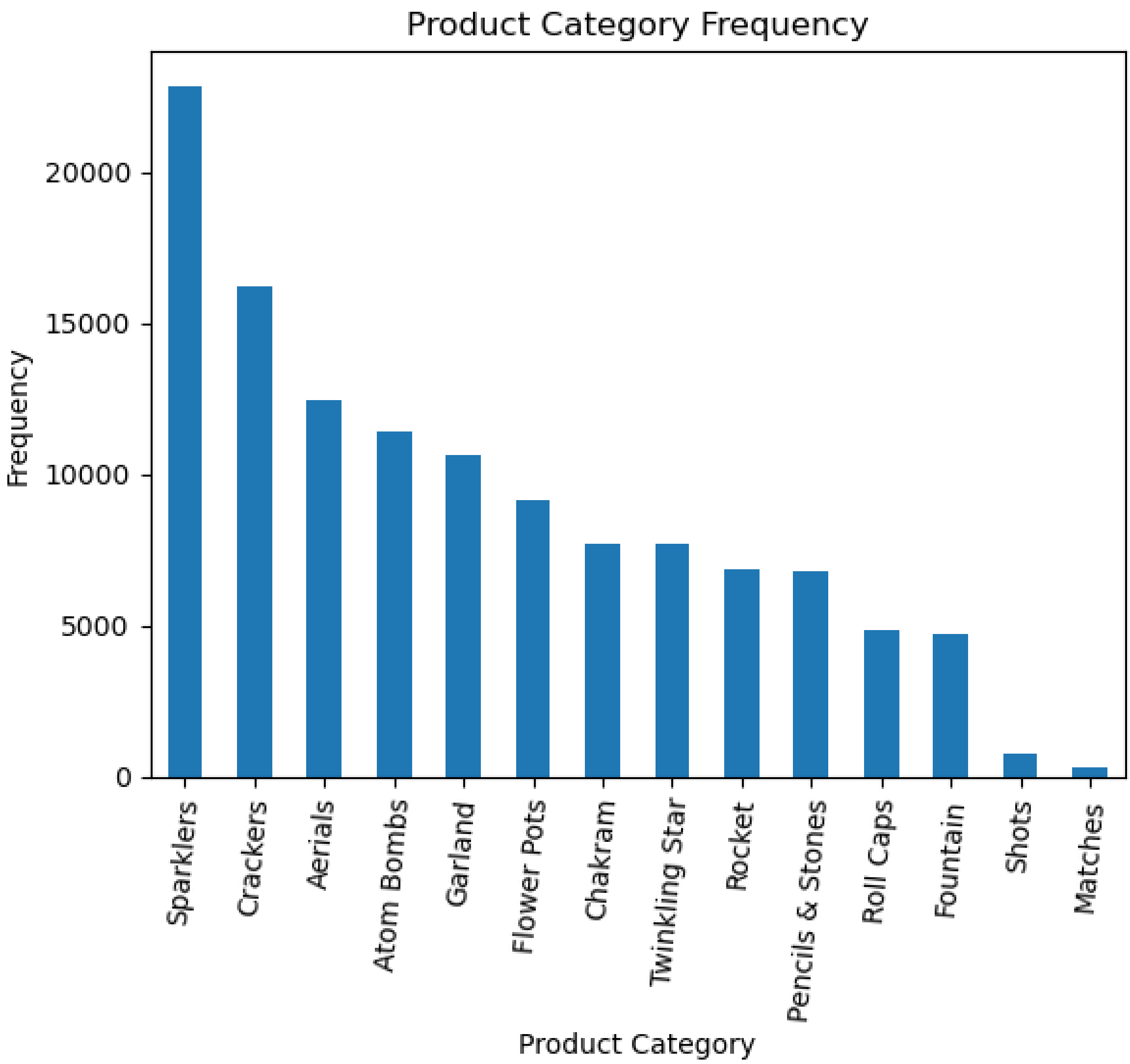
Out[13]:

	billId	sequenceNo	skuId	quantity	rate	amount	productName	productCategory	da
0	29029	1	70	2	670	1340	Aerials Green Boom 3 per Box	Aerials	201 0
1	29029	2	60	1	405	405	Knife 100 Shots	Shots	201 0
2	29029	3	9	2	175	350	Coronation Candle 10 Per Box	Pencils & Stones	201 0
3	29029	4	34	3	60	180	Bijili Crackers Red 50 Per Box	Crackers	201 0
4	29029	5	62	5	85	425	Roll Caps	Roll Caps	201 0

In [14]:

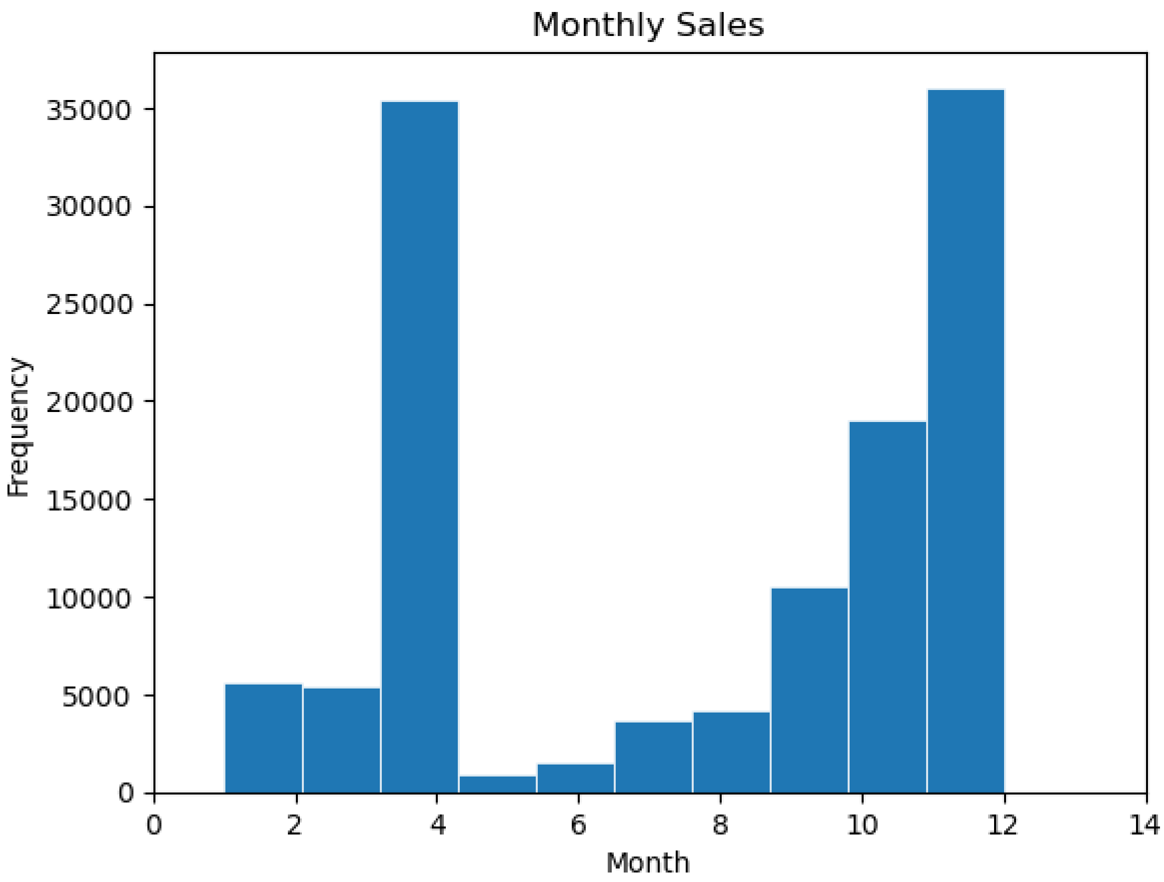
```
# plot Product Category Frequency

train['productCategory'].value_counts().plot(kind='bar',ylabel='Frequency',t
plt.xlabel("Product Category")
```

Out[14]: Text(0.5, 0, 'Product Category')



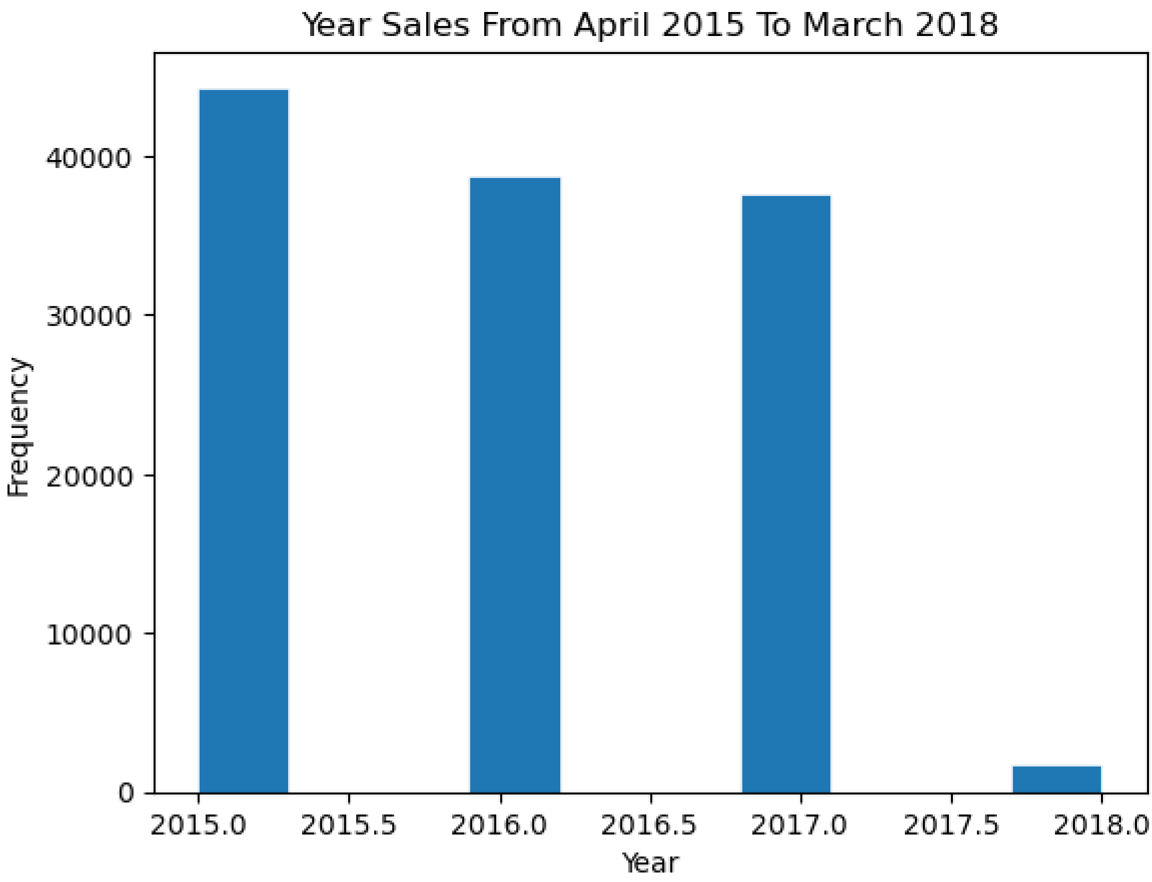
In [15]:

```
# Plot monthly sales

x = train.month
# plot:
fig, ax = plt.subplots()

ax.hist(x, bins=10, linewidth=0.5, edgecolor="white")

ax.set(xlim=(0, 14))
ax.set_title('Monthly Sales')
ax.set_xlabel('Month')
ax.set_ylabel('Frequency')
plt.show()
```



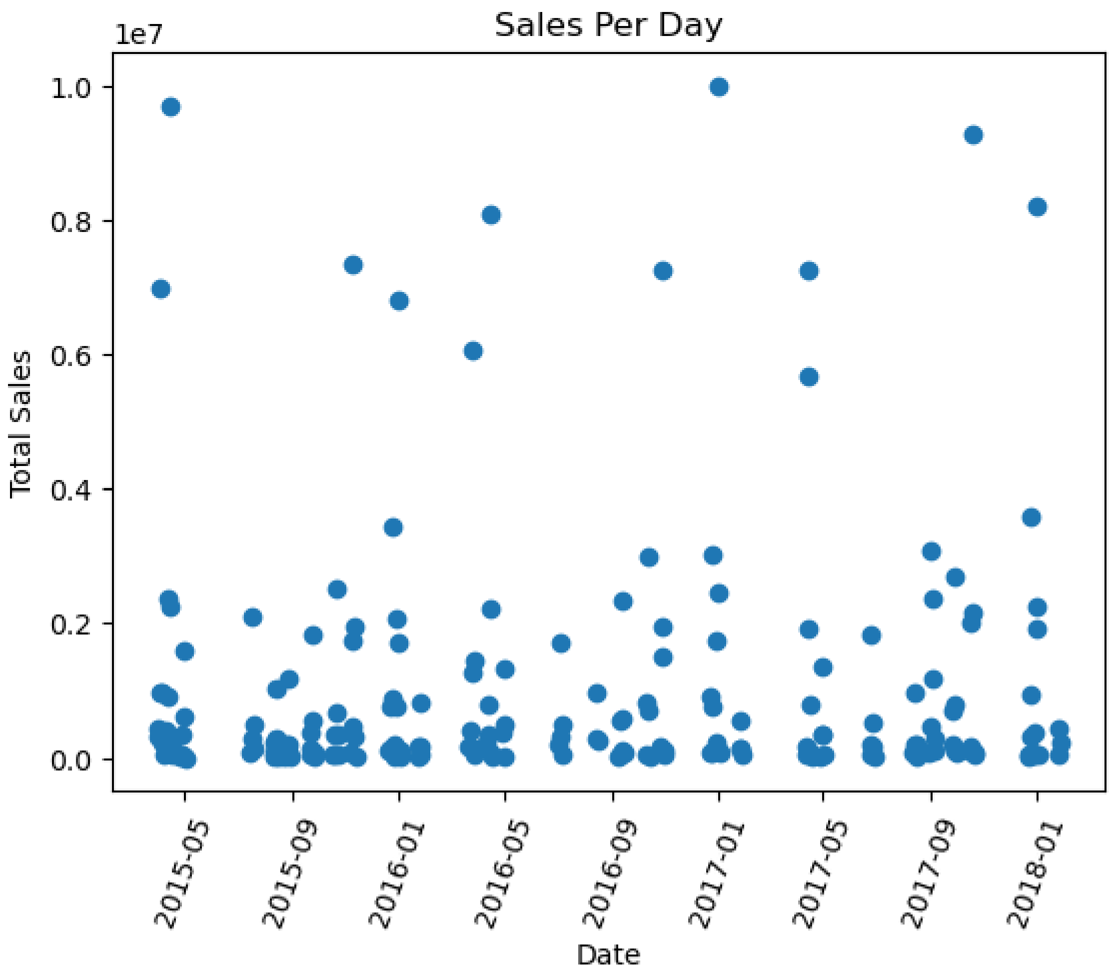
In [16]:

```
# Plot yearly sales from 2015 April to March 2018
x = train.year
# plot:
fig, ax = plt.subplots()

ax.hist(x, bins=10, linewidth=0.5, edgecolor="white")

ax.set_title('Year Sales From April 2015 To March 2018')
ax.set_xlabel('Year')
ax.set_ylabel('Frequency')
plt.show()
```



In [17]:
```
yearly_train = train.copy()
date_tain = yearly_train.date.unique()
yearly_train = pd.DataFrame(yearly_train.groupby(['date'])['total'].sum())
yearly_train.columns = ["total"]
```

In [18]:
```
# Plot total sales against date

fig, ax = plt.subplots()
ax.scatter(date_tain, yearly_train['total'])
plt.xticks(rotation=70)
ax.set_xlabel('Date')
ax.set_ylabel('Total Sales')
ax.set_title("Sales Per Day")
plt.show()
```



In [19]:
```
# check for any null values in the data frame
train.isnull().sum()
```

Out[19]:
```
billId             0
sequenceNo         0
skuId              0
quantity           0
rate               0
amount             0
productName        0
productCategory    0
date               0
retailerId         0
total              0
year               0
month              0
dtype: int64
```

In [20]:
```
# find the outliers
for x in range(0,101,10) :
    print(f'{x}th percentile value for quantity is {np.percentile(train["amc
for x in range(0,101,10) :
    print(f'{x}th percentile value for rate is {np.percentile(train["rate"],
```

```
0th percentile value for quantity is 26.0
10th percentile value for quantity is 112.0
20th percentile value for quantity is 171.0
30th percentile value for quantity is 225.0
40th percentile value for quantity is 270.0
50th percentile value for quantity is 300.0
60th percentile value for quantity is 340.0
70th percentile value for quantity is 400.0
80th percentile value for quantity is 450.0
90th percentile value for quantity is 556.0
100th percentile value for quantity is 1728.0
0th percentile value for rate is 26.0
10th percentile value for rate is 49.0
20th percentile value for rate is 68.0
30th percentile value for rate is 81.0
40th percentile value for rate is 90.0
50th percentile value for rate is 110.0
60th percentile value for rate is 116.0
70th percentile value for rate is 134.0
80th percentile value for rate is 171.0
90th percentile value for rate is 280.0
100th percentile value for rate is 864.0
```

In [21]:

```
# remove the outliers for
train = train[(train['rate']<train['rate'].quantile(0.98))]
train = train[(train['rate']<train['amount'].quantile(0.98))]
train.head()
```

Out[21]:

	billId	sequenceNo	skuId	quantity	rate	amount	productName	productCategory	da
1	29029	2	60	1	405	405	Knife 100 Shots	Shots	201 0
2	29029	3	9	2	175	350	Coronation Candle 10 Per Box	Pencils & Stones	201 0
3	29029	4	34	3	60	180	Bijili Crackers Red 50 Per Box	Crackers	201 0
4	29029	5	62	5	85	425	Roll Caps	Roll Caps	201 0
5	29029	6	12	4	110	440	Flower Pots Small 10 Per Box	Flower Pots	201 0

Methods

Decision Tree / Classification And Regression Trees (CART):

Decision Tree is a supervised learning algorithm used for predictive analysis on tabular data. We are using a variant of Decision Tree called Classification And Regression Trees (CART). The hypothesis is that CART can be a good for the data in question.

Random Forest:

Random Forest is a popular supervised machine learning algorithm and is an ensemble method used for classification and regression (for numeric data) tasks. Since the target variable is a numeric value, an ensemble method might improve the prediction over a CART.

Linear Regression:

Linear Regression tries to fit a line which minimises the distance between the a set of points. This was an attempt to see whether a simple Linear Regression will suffice for prediction compared to ensemble methods.

Gradient Boosting:

Gradient boosting is also a popular supervised machine learning algorithm and is an ensemble method for regression and classification problems. It aggregates an ensemble of weak individual models which are typically decision tress, to obtain a more accurate final model. This was also an attempt to see whether Gradient Boosting Regressor improve the prediction over the Random Forest.

Model Development

In [22]:

```
# preparing the training and testing set
feature,label = train[['skuId','quantity']], train['amount']
X_train, X_test, y_train, y_test = train_test_split(feature,label, test_size
```

In [ ]:

```
# Method to evaluate the performance of the models
def evaluateModel(model, X_test, y_test) :
    modelPrediction = model.predict(X_test)
    modelError = abs(modelPrediction - y_test)
    modelMAP = 100 * np.mean(modelError/y_test)
    modelMSE = metrics.mean_squared_error(y_test, modelPrediction)
    modelAccuracy = 100 - modelMAP
    print('Model Performance')
    print('Average Error: {:0.4f} degrees'.format(np.mean(modelError)))
    print('Mean Square Error :{:0.2f}%'.format(modelMSE) )
    print('Accuracy: {:0.2f}%'.format(modelAccuracy))

    return modelAccuracy
```

Random Forest

In [24]:

```
randomModel = RandomForestRegressor(random_state = 42, n_estimators=1000)
randomModel.fit(X_train,y_train)
# pprint used to prettify a data structure
pprint(randomModel.get_params())
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': None,
 'max_features': 1.0,
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 1000,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

Feature Importance

In [39]:

```
# generate the importance
imp = randomModel.feature_importances_
# summarizing feature importance
for f,s in enumerate(imp):
 print('Feature: %0d, Score: %.5f' % (f,s))
```

```
Feature: 0, Score: 0.73519
Feature: 1, Score: 0.26481
```

Decision Tree

In [26]:

```
decisionTreeModel = DecisionTreeRegressor(random_state=0, max_depth=10)
decisionTreeModel = decisionTreeModel.fit(X_train, y_train)
```

Gradient Boosting

In [27]:

```
parameters = {
    "n_estimators": 200,
    "max_depth": 10,
    "min_samples_split": 10,
    "learning_rate": 0.01,
    "loss": "squared_error",
}

gbRegModel = ensemble.GradientBoostingRegressor(**parameters)
gbRegModel.fit(X_train, y_train)
```

Out[27]:

```
▾ GradientBoostingRegressor
GradientBoostingRegressor(learning_rate=0.01, max_depth=10,
                          min_samples_split=10, n_estimators=200)
```

Linear Regression

In [28]:

```
linearRegModel = LinearRegression()
linearRegModel.fit(X_train, y_train)
```

Out[28]:

```
▾ LinearRegression
LinearRegression()
```

Result

In [29]:

```
# evaluating the Random Forest Regressor
rfAccuracy = evaluateModel(randomModel, X_test, y_test)
```

```
Model Performance
Average Error: 4.7275 degrees
Mean Square Error :52.64%
Accuracy: 98.43%
```

In [30]:

```
## evaluating the Decision Tree Regressor
dtAccuracy = evaluateModel(decisionTreeModel, X_test, y_test)
```

```
Model Performance
Average Error: 13.0940 degrees
Mean Square Error :735.69%
Accuracy: 95.40%
```

In [31]:

```
## evaluating the Gradient Boosting Regressor
gbAccuracy = evaluateModel(gbRegModel, X_test, y_test)
```

```
Model Performance
Average Error: 18.3251 degrees
Mean Square Error :532.79%
Accuracy: 90.51%
```

In [32]:

```
## evaluating the Linear Regression Model
lrAccuracy = evaluateModel(linearRegModel, X_test, y_test)
```

```
Model Performance
Average Error: 115.1914 degrees
Mean Square Error :21823.25%
Accuracy: 42.82%
```

Prediction

Predicting the models with sample test data

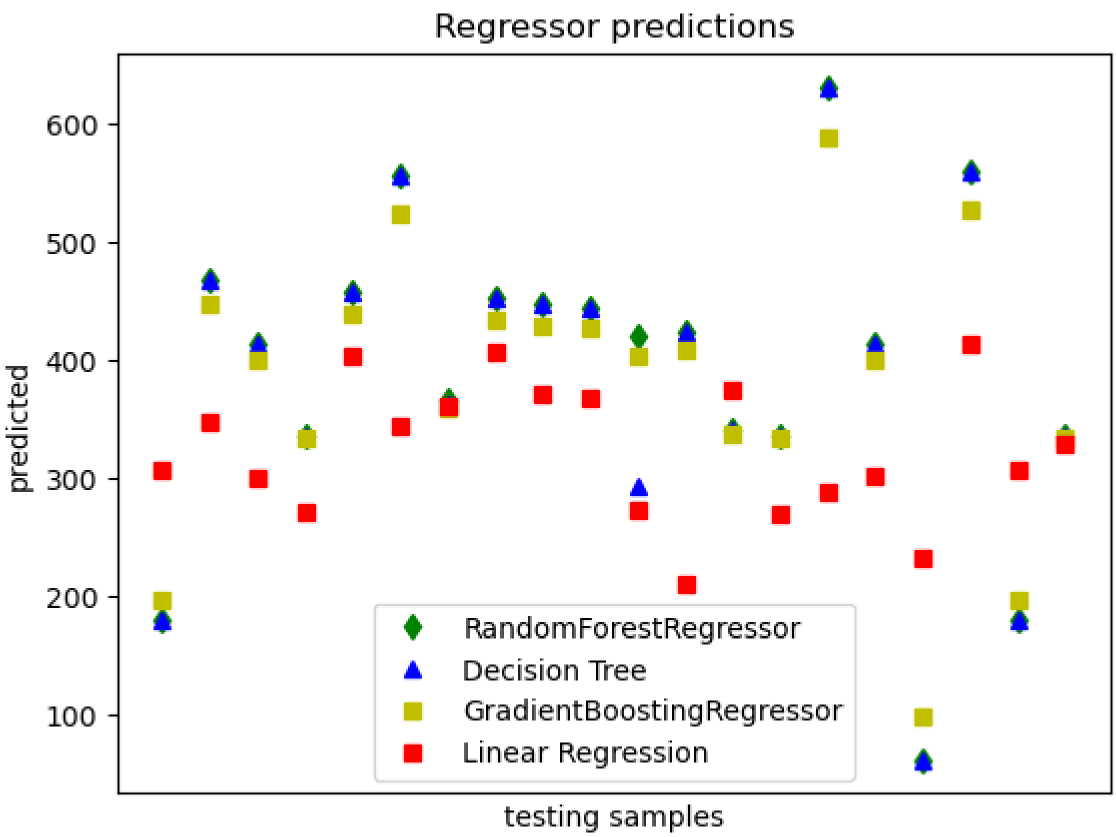
In [33]:

```
xt = X_test[:20]

pred1 = randomModel.predict(xt)
pred2 = decisionTreeModel.predict(xt)
pred3 = gbRegModel.predict(xt)
pred4 = linearRegModel.predict(xt)
print(pred1)
print(pred2)
print(pred3)
print(pred4)
```

```
[180.         467.65100991 413.08029906 336.02757441 457.75890973
 555.85714045 366.70999131 452.48146783 447.14360114 443.56849229
 420.         423.84621828 340.         336.33451558 631.2050657
 413.49768654  61.61817377 559.98565151 180.         336.01561343]
[180.         467.64971751 413.28813559 336.0242915  457.7690583
 555.92286501 361.92954828 452.482066   447.14129244 443.56985605
 293.62868118 423.84922395 340.         336.34081902 631.20604915
 413.53667954  61.61570248 559.96661102 180.         336.01298701]
[195.8340915  447.24887413 399.99167463 333.25857989 438.69202242
 523.69521402 360.02696225 434.11337963 429.48816118 426.39522467
 403.62009935 409.31675644 336.70162426 333.53287722 588.89198163
 400.38588338  97.82595194 527.19718025 195.8340915  333.24878998]
[306.91949223 348.2514923  299.38204655 270.43299296 403.0432168
 344.48276945 360.63443877 407.35032861 371.9406073  367.0951065
 273.12493785 210.79576766 374.09416321 269.89460398 287.66144023
 302.07399143 232.86971573 414.34938532 306.91949223 329.53182928]
```

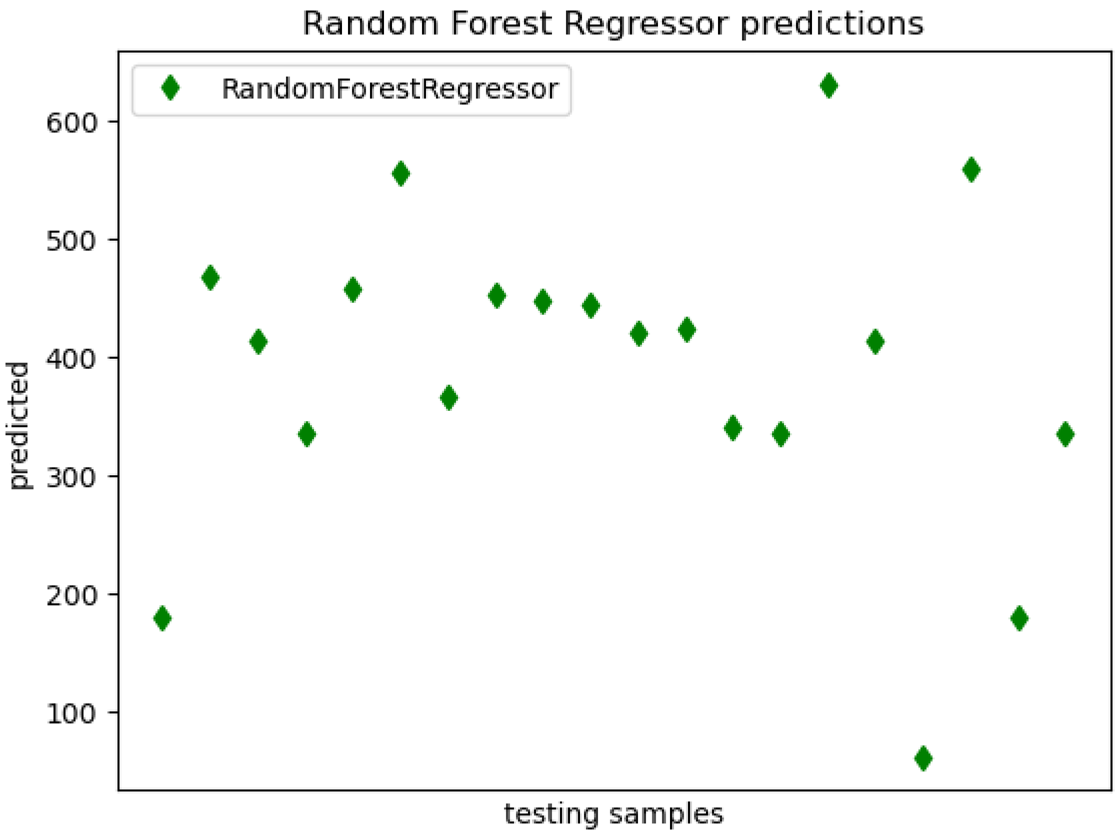
In [34]:

```
fig, ax = plt.subplots()
ax.plot(pred1, "gd", label="RandomForestRegressor")
ax.plot(pred2, "b^", label="Decision Tree")
ax.plot(pred3, "ys", label="GradientBoostingRegressor")
ax.plot(pred4, "rs", label="Linear Regression")
ax.tick_params(axis="x", which="both", bottom=False, top=False, labelbottom=
ax.set_ylabel("predicted")
ax.set_xlabel("testing samples")
ax.legend(loc="best")
ax.set_title("Regressor predictions")
plt.show()
```

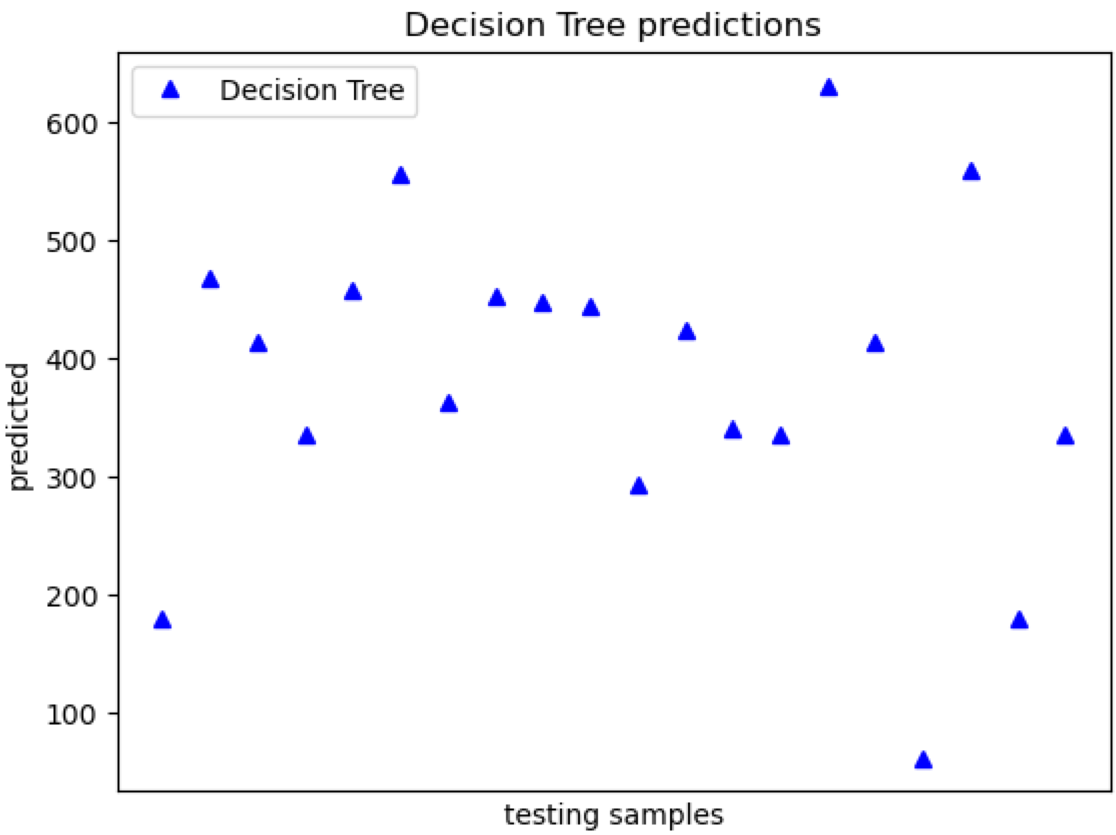


In [35]:

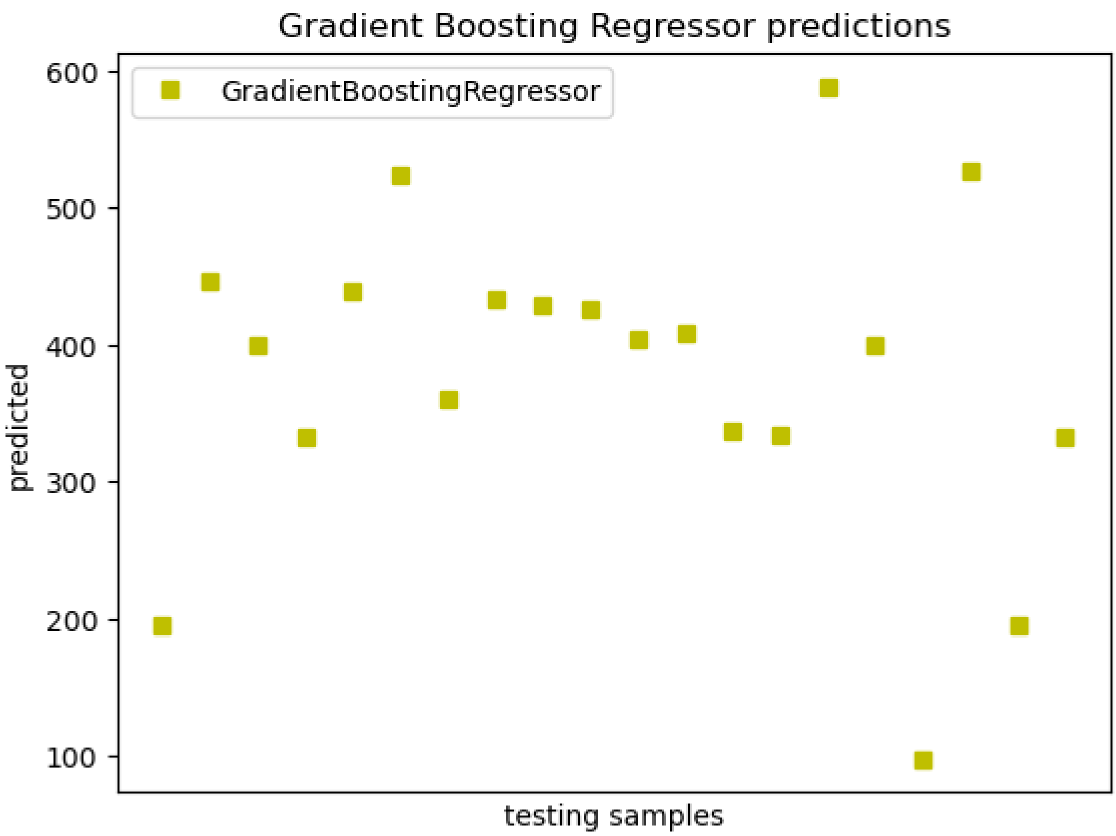
```
fig, ax = plt.subplots()
ax.plot(pred1, "gd", label="RandomForestRegressor")
ax.tick_params(axis="x", which="both", bottom=False, top=False, labelbottom=
ax.set_ylabel("predicted")
ax.set_xlabel("testing samples")
ax.legend(loc="best")
ax.set_title("Random Forest Regressor predictions")
plt.show()
```



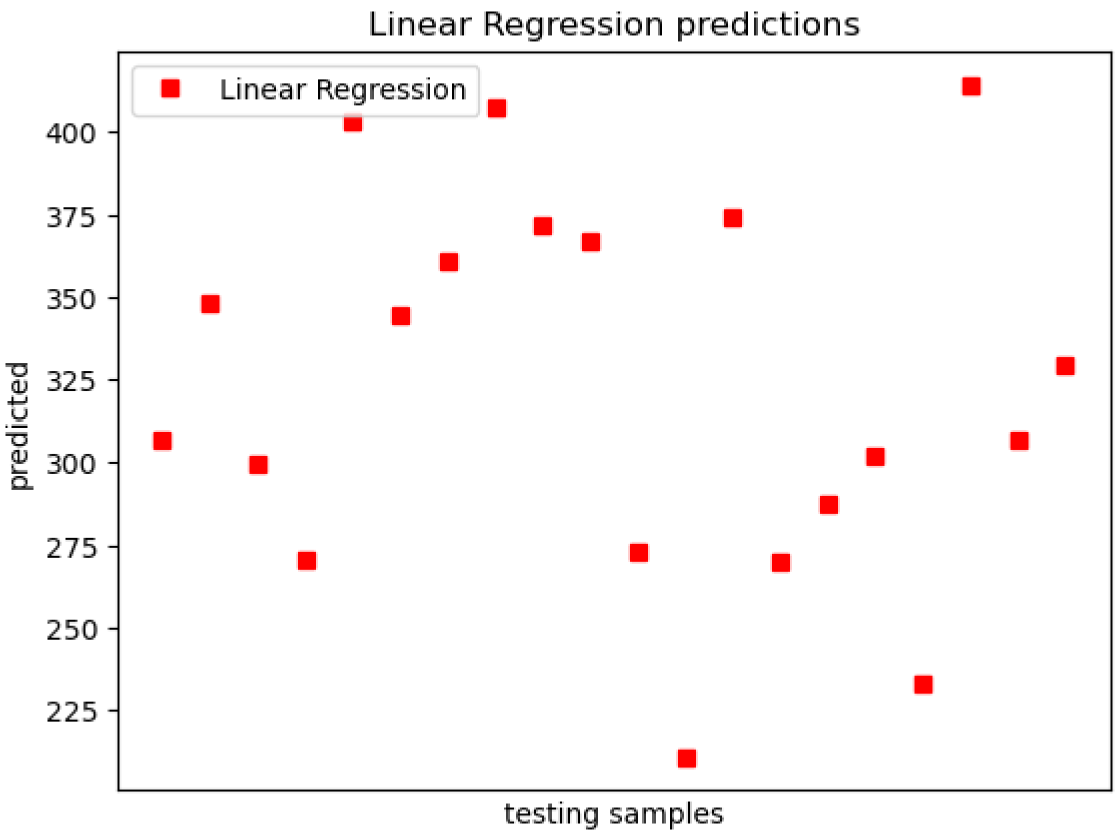
In [36]:
```
fig, ax = plt.subplots()
ax.plot(pred2, "b^", label="Decision Tree")
ax.tick_params(axis="x", which="both", bottom=False, top=False, labelbottom=
ax.set_ylabel("predicted")
ax.set_xlabel("testing samples")
ax.legend(loc="best")
ax.set_title("Decision Tree predictions")

plt.show()
```



In [37]:
```
fig, ax = plt.subplots()
ax.plot(pred3, "ys", label="GradientBoostingRegressor")
ax.tick_params(axis="x", which="both", bottom=False, top=False, labelbottom=
ax.set_ylabel("predicted")
ax.set_xlabel("testing samples")
ax.legend(loc="best")
ax.set_title("Gradient Boosting Regressor predictions")
plt.show()
```



In [38]:
```
fig, ax = plt.subplots()
ax.plot(pred4, "rs", label="Linear Regression")
ax.tick_params(axis="x", which="both", bottom=False, top=False, labelbottom=
ax.set_ylabel("predicted")
ax.set_xlabel("testing samples")
ax.legend(loc="best")
ax.set_title("Linear Regression predictions")
plt.show()
```



Comparing the performance of the models

Model	Average Error	Mean Square Error	Accuracy
Random Forest	4.7275 degrees	52.64%	98.43%
Decision Tree	13.0940 degrees	735.69%	95.40%
Gradient Boosting	18.3251 degrees	532.79%	90.51%
Linear Regression	115.1914 degrees	21823.25%	42.82%

Conculsion

For predicting the sales data, we applied different supervised learning techniques. Of these, Random Forest and Decision Tree gave the best result on the prediction. Gradient Boosting prediction accuracy was 90%. Linear Regression did not perform well.