In [1]:
```
#1
import pandas as pd
import numpy as np

data = pd.read_excel("Sales_analysis_Dataset.xlsx")
print(data.head())
```

```
      TV  Radio  Newspaper  Sales
0  230.1   37.8       69.2   22.1
1   44.5   39.3       45.1   10.4
2   17.2   45.9       69.3   12.0
3  151.5   41.3       58.5   16.5
4  180.8   10.8       58.4   17.9
```

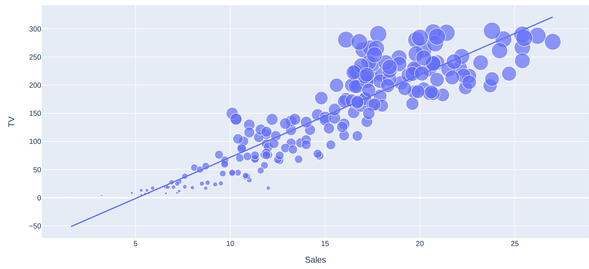
1> Notes: Some times you will see error when you will use "data = pd.read_csv()" commend .This time you need to import openpyxl library using commend -->pip install openpyxl and need to use "data =pd.read_excel()"

In [2]:
```
#2
print(data.isnull().sum())
```

```
TV           0
Radio        0
Newspaper    0
Sales        0
dtype: int64
```

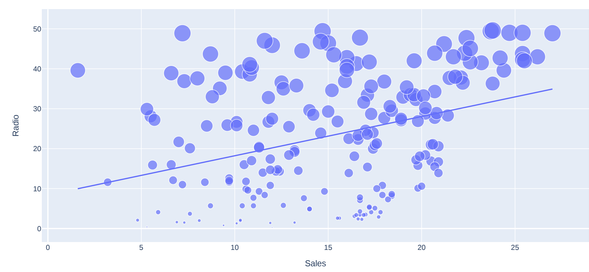
2> Notes: Need to check whether any column have any null/NAN value .This will calculate the sum of nulls/NAN values for each column.Here in this dataset doesn't have any null values.

In [3]:
```
#3 Ploting the graph of amount of sales(Product count) increase based on money invested for Tv advertisement

import plotly.express as px

figure=px.scatter(data_frame=data,x="Sales",y="TV",size="TV",trendline="ols")

figure.show()
```

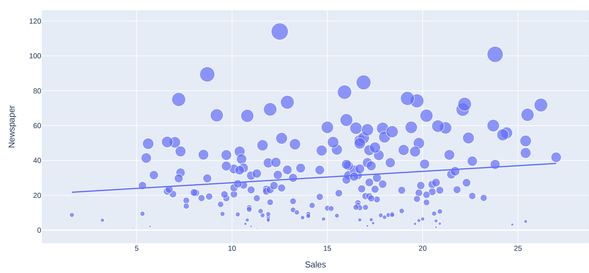


3> Notes : OLS stands for Ordinary Least Squares, which is a commonly used method for estimating the parameters of a linear regression model.When you specify "trendline='ols'", it suggests that you want to use OLS to calculate the trendline for your data. This means that you want to find the best-fitting straight line that represents the relationship between the independent variable (X) and the dependent variable (Y).

In [4]:
```
#4 Ploting the graph of amount of sales(Product count) increase based on money invested for Radio advertisement

figure=px.scatter(data_frame=data ,x="Sales",y="Radio",size="Radio",trendline="ols")

figure.show()
```



In [5]:
```
#5 Ploting the graph of amount of sales(Product count) increase based on money invested for newspaper advertisement

figure =px.scatter(data_frame=data,x="Sales",y="Newspaper",size="Newspaper",trendline="ols")
figure.show()
```



5> Conclusion: Out of all the amount spent on advertising on various platforms, I can see that the amount spent on advertising the product on TV results in more sales of the product.

In [21]:
```
#6 Now let's have a look at the correlation of all the columns with the sales column.

correlation=data.corr()
print(correlation["Sales"].sort_values(ascending =False))
```

```
Sales        1.000000
TV           0.901208
Radio        0.349631
Newspaper    0.157960
Name: Sales, dtype: float64
```

6> Notes :Here the numbers shows that from TV advertisement we able to get maximum sales ,then Radio advertisement ,then newpapers.sort_values(ascending =False)) --> arranging the data in a descending order

---------- Future Sales Prediction Model----------

Now in this section, I will train a machine learning model to predict the future sales of a product. But before I train the model, let's split the data into training and test sets:

In [29]:
```
#i)

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

x = np.array(data.drop(["Sales"],axis=1))
y=np.array(data["Sales"])

# Assuming you have your features and labels in x and y respectively
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2, random_state=42)
```

Notes :

i) -->x and y are the input features and corresponding labels, respectively, that you want to split into training and testing sets.

-->Test_size=0.2 indicates that you want to allocate 20% of the data for testing, while the remaining 80% will be used for training.

-->random_state=42 sets the random seed value to 42, ensuring that the same random split is generated each time you run the code. This helps with reproducibility of the results.

-->The train_test_split function returns four variables in the following order: x_train, x_test, y_train, and y_test. These variables will contain the training and testing subsets of the input features (x) and labels (y), respectively.

-->By splitting your data into separate training and testing sets, you can train your model on the training data and then evaluate its performance on the unseen testing data. This helps you assess how well your model generalizes to new, unseen examples.

In [30]:
```
#ii) Now let's train the model to predict future sales

model =LinearRegression()
model.fit(xtrain,ytrain)
print(model.score(xtest,ytest))
```

```
0.9059011844150826
```

Notes:

ii)

-->The code snippet you provided is using scikit-learn's `LinearRegression` model to fit a linear regression on the training data (`xtrain` , `ytrain`). After training the model, it is evaluating its performance on the test data (`xtest` , `ytest`) using the `score` method and printing the result.

-->The `score` method of the `LinearRegression` model in scikit-learn returns the coefficient of determination, denoted as R-squared. R-squared is a statistical measure that represents the proportion of the variance in the dependent variable (y) that can be explained by the independent variables (x) in the model. It provides an indication of how well the linear regression model fits the data.

-->The `score` method computes the R-squared value for the given test data and the corresponding true labels. R-squared values range from 0 to 1, where a value of 1 indicates a perfect fit, and a value of 0 indicates that the model does not explain any of the variance in the target variable beyond the mean.

-->By printing `model.score(xtest, ytest)` , you will get the R-squared value as the output, which represents the model's performance on the test data. The closer the R-squared value is to 1, the better the model's fit.

--> It's important to ensure that the variables (`xtrain` , `ytrain` , `xtest` , `ytest`) are appropriately preprocessed and formatted before using them with the `LinearRegression` model.

In [32]:
```
#iii) Now let's input values into the model according to the features,
#we have used to train it and predict how many units of the product can be sold,
#based on the amount spent on its advertising on various platforms:

#features = [[TV, Radio, Newspaper]]

features=np.array([[230.21,36.8,69.2]])
print(model.predict(features))
```

```
[21.27759094]
```

Summary :If we invest 230.21 $inTvadvertisement, 36.8$ in Radio advertisement and 69.2$ in Newspaper advertisement the approx number of product sale will be 21.27

In [ ]: