Case study -Polynomial Regression.

This case study will focus on finding a polynomial model for a data set that is described in *"An Introduction to Statistical Learning with Applications in Python."* A must read book for anybody who wants to learn about Data Analysis. A free pdf download of the book can be found here. Or if the link has changed search for ISLR free pdf download. The data is an Advertising data set. It shows the relationship in sales, in thousands of units of a particular product as a function of TV, radio, and newspaper advertisement, money spend in thousands of dollars, for 200 different markets.

Problem Introduction.

From past analysis it is well known that there is a positive relationship between advertisement and sale of a product, the more is advertised, the more gets sold. However, the relationship is not linear. And as we increase advertisement the slope in sale decreases. Eventually we get to a point from where no matter how much is advertised, the sales the does not change, most likely because, almost everybody who wants to buy the product already bought it.

The objective of this study is to find the existence of relationship between advertisement money spend and units sold. Also, since there are several channels of advertisement. The study wants to find out which one is more effective or which combination of ad forms are more effective.

Data Exploration.

df = pd.read_csv("Advertising.csv")

df.head()

The five rows of the data set.

	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	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

frdf.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   TV         200 non-null    float64
 1   radio      200 non-null    float64
 2   newspaper  200 non-null    float64
 3   sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
```

```
There are 200 rows with all numerical data.
```

df.describe()

	TV	radio	newspaper	sales
count	200.000000	200.000000	200.000000	200.000000
mean	147.042500	23.264000	30.554000	14.022500
std	85.854236	14.846809	21.778621	5.217457
min	0.700000	0.000000	0.300000	1.600000
25%	74.375000	9.975000	12.750000	10.375000
50%	149.750000	22.900000	25.750000	12.900000
75%	218.825000	36.525000	45.100000	17.400000
max	296.400000	49.600000	114.000000	27.000000

```
df.corr()
```

```
Checking if there are some correlation between coefficients. In this case we are
looking for a Correlation between different medium of advertisement (TV, Radio,
Newspaper) and Sales.
```

	TV	radio	newspaper	sales
TV	1.000000	0.054809	0.056648	0.782224
radio	0.054809	1.000000	0.354104	0.576223
newspaper	0.056648	0.354104	1.000000	0.228299
sales	0.782224	0.576223	0.228299	1.000000

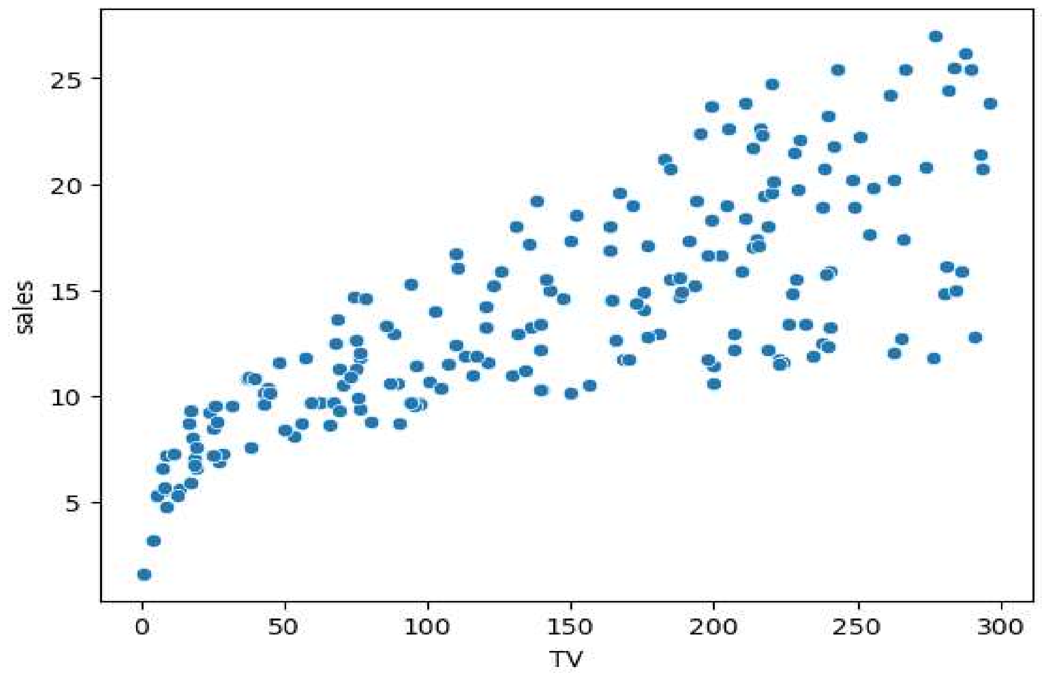
Looks like There is a good correlation between TV and Sales (0.78). Between Radio and Sales is 0.57. Not pretty good but still there is some. The correlation coefficient between Newspaper and Sales is 0.23. Basically non existent.

Data visualizations.

1. TV vs Sales.

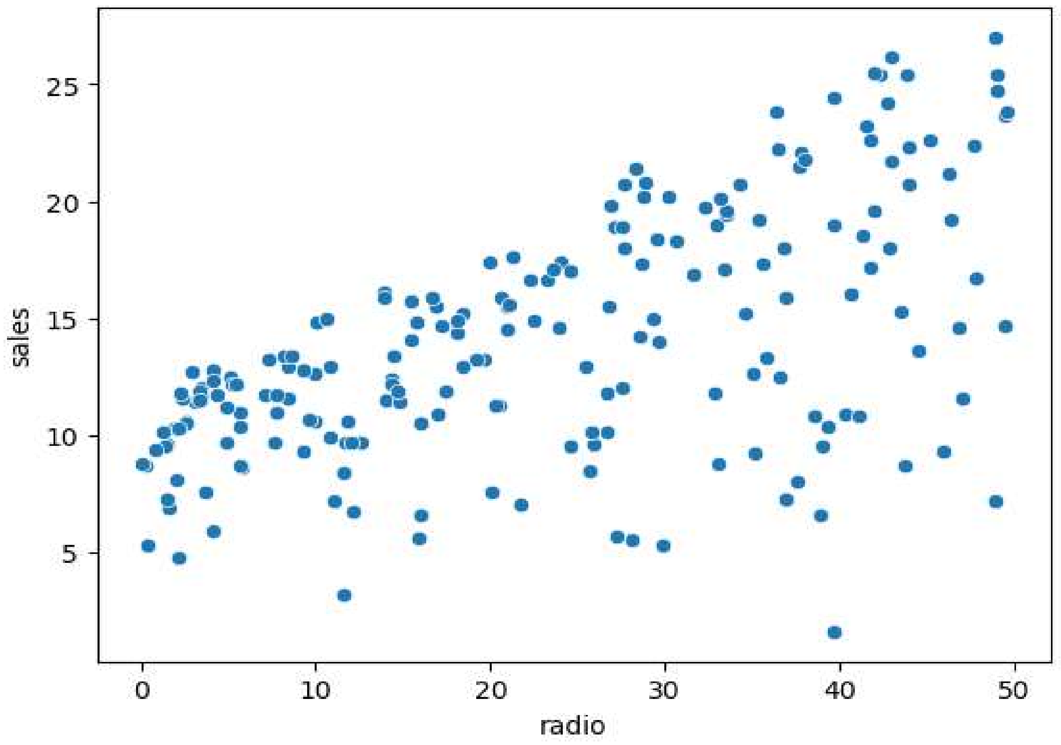
sns.scatterplot(x='TV',y='sales',data=df)

We can see clearly there is a positive relationship between Money spend on TV and Generated Sales. As we increase Money spend generated sales increases. And the variance increases as money spend increase.



2. Radio vs Sales.

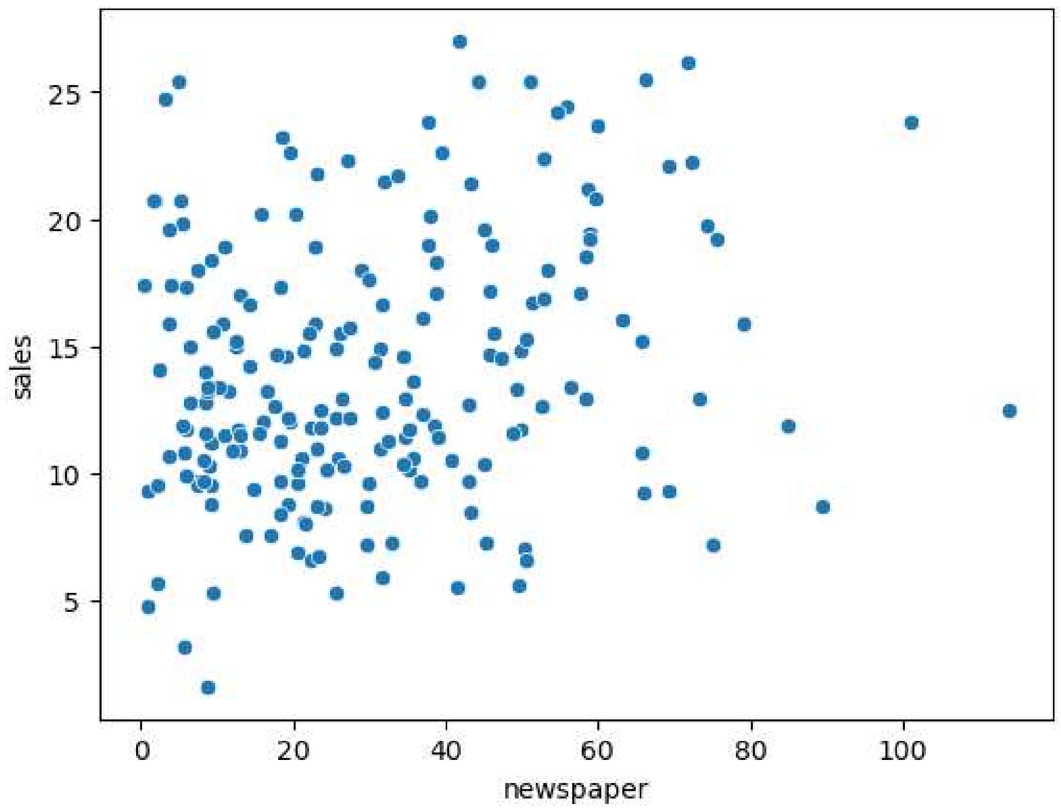
sns.scatterplot(x='radio',y='sales',data=df)



As the coefficient of correlation suggested not a well defined trend. Although, we can say there is some positive relationship. The variance is very wide.

3. Newspaper vs Sales.

sns.scatterplot(x='newspaper',y='sales',data=df)



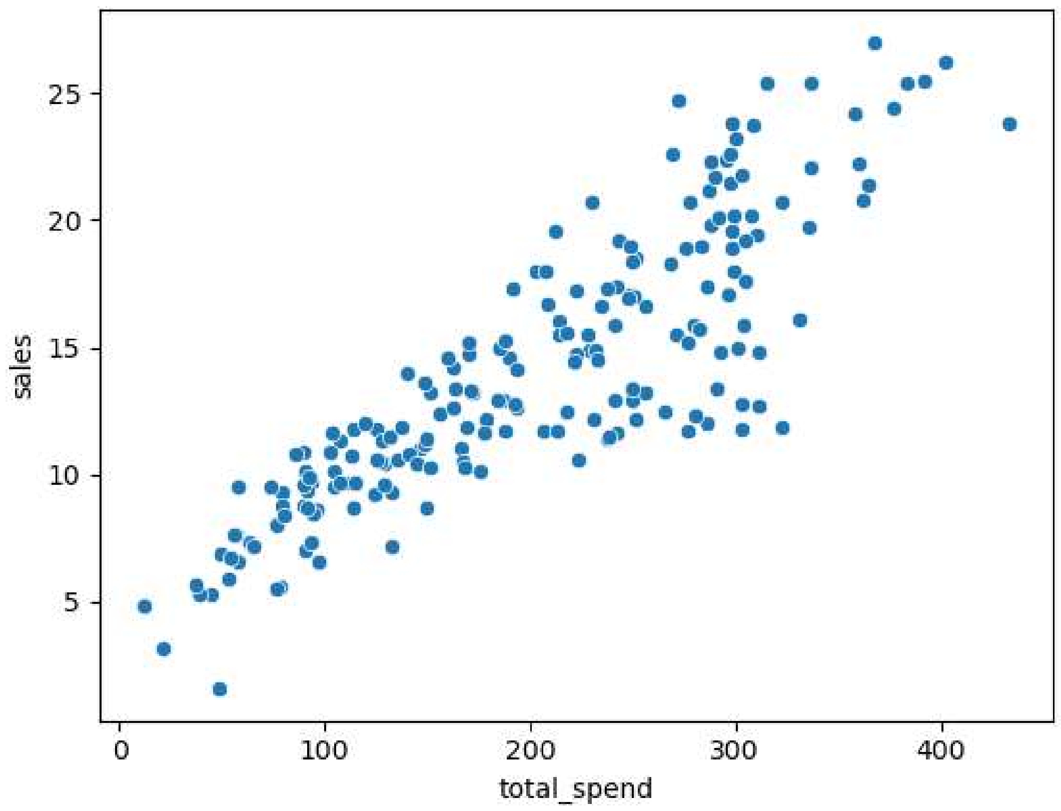
In [ ]:

There is no clear relationship between Mnoney spend in newspaper and generated sales.

Finally let’s group all of mediums and compare to sales.

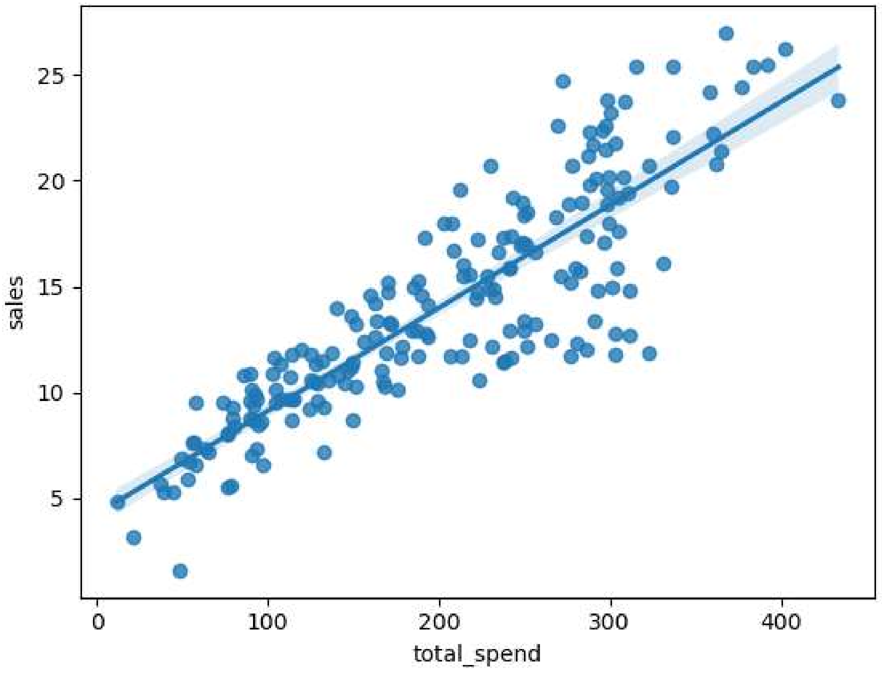
df['total_spend'] = df['TV'] + df['radio'] + df['newspaper']

sns.scatterplot(x='total_spend',y='sales',data=df)



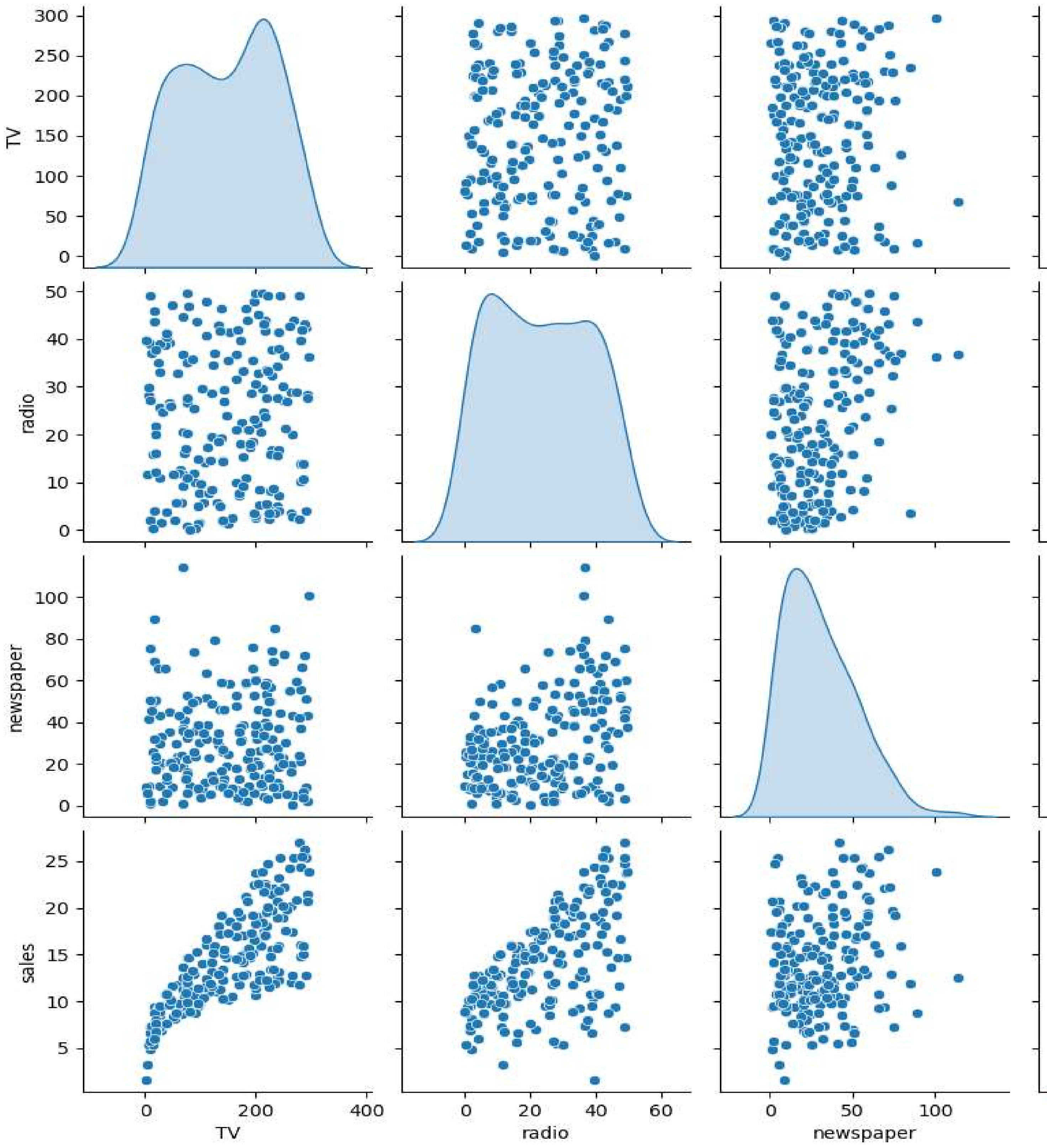
Clearly there is some relationship. We can even draw a line on it.

sns.regplot(x='total_spend',y='sales',data=df)

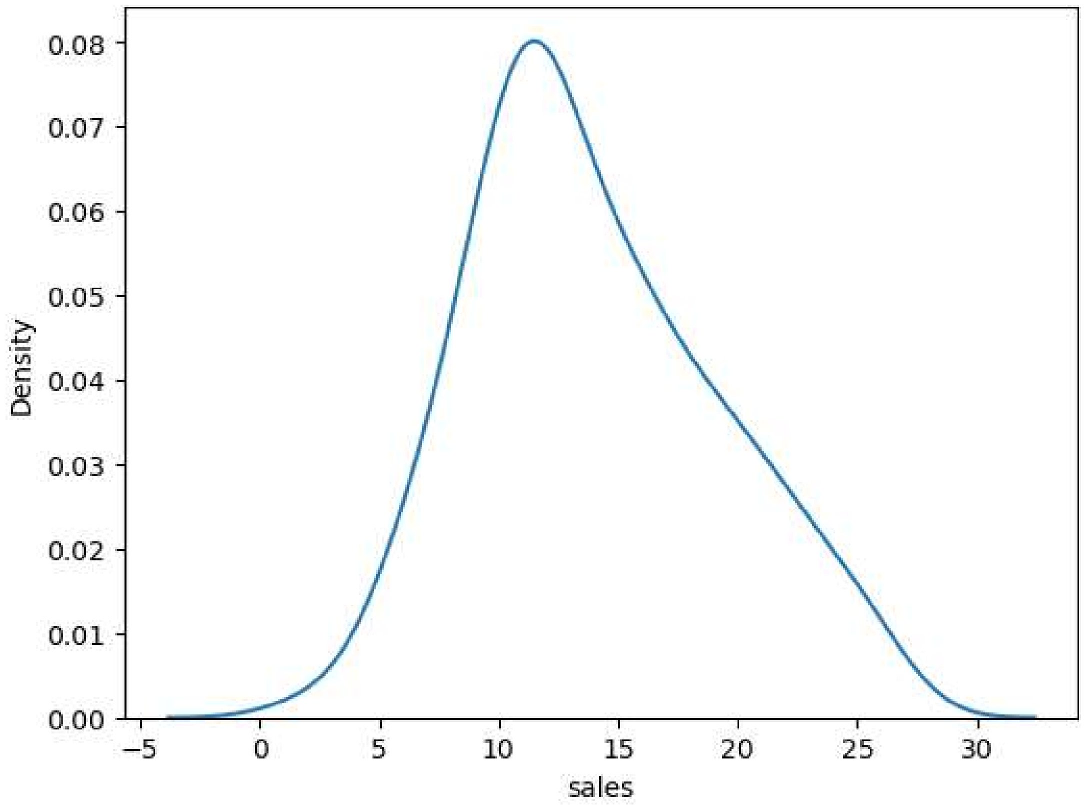


In [ ]:

sns.pairplot(df,diag_kind='kde')



sns.kdeplot(df['sales'])



We will try to find a relationship between money spend in TV, Radio and Newspaper and generated Sales. Sales is the label we are trying to predict.

X = df.drop('sales',axis=1)

y = df['sales']

from sklearn.preprocessing import PolynomialFeatures
polynomial_converter = PolynomialFeatures(degree=2,include_bias=False)

poly_features = polynomial_converter.fit_transform(X)

The correlation between TV and sales is strong, between newspaper and sales is week. How about if we combine TV and newspaper together and then check if there is a stronger correlation. This is what polynomial model of higher order will try to figure out. If we check the shape of the data set, now it will have nine coefficients.

poly_features.shape

(200, 9)

```
poly_features[0]
```

```
array([2.301000e+02, 3.780000e+01, 6.920000e+01, 5.294601e+04,
       8.697780e+03, 1.592292e+04, 1.428840e+03, 2.615760e+03,
       4.788640e+03])
```

In order to evaluate model's performance we need to set aside portion of the data set. Once we build a model on specific portion of the data, which we call training set, then we will test the performance on the unseen testing set.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(poly_features, y, test_size=0.3,
random_state=42)
```

Now we are creating an instance of a specific model.

```
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)
```

Fitting the training data into the model.

```
model.fit(X_train,y_train)
```

We have finished with building the model. Now we need to test it's performance.
First, we need to predict the sales amount using the model. As input value we will use the available data left for testing.

```
test_predictions = model.predict(X_test)
```

Second, we need to compare the predicted sales(test_prediction) to the real sales amount that have already occurred. And, they are available in the testing data set that was left aside(y_test).

```
from sklearn.metrics import mean_absolute_error,mean_squared_error
```

```
MAE = mean_absolute_error(y_test,test_predictions)
MSE = mean_squared_error(y_test,test_predictions)
RMSE = np.sqrt(MSE)
```

```
MAE
```

```
0.5905974833808129
```

```
MSE
```

```
0.5231944949055536
```

RMSE

```
0.7233218473857634
```

These are the error metrics using polynomial model with degree=2. Can we improve the performance of the model? How about if we use higher order. Maybe dgree=5 or degree=10.

Let’s try to build a function that will perform at once all the steps that we have performed for models with different degrees.

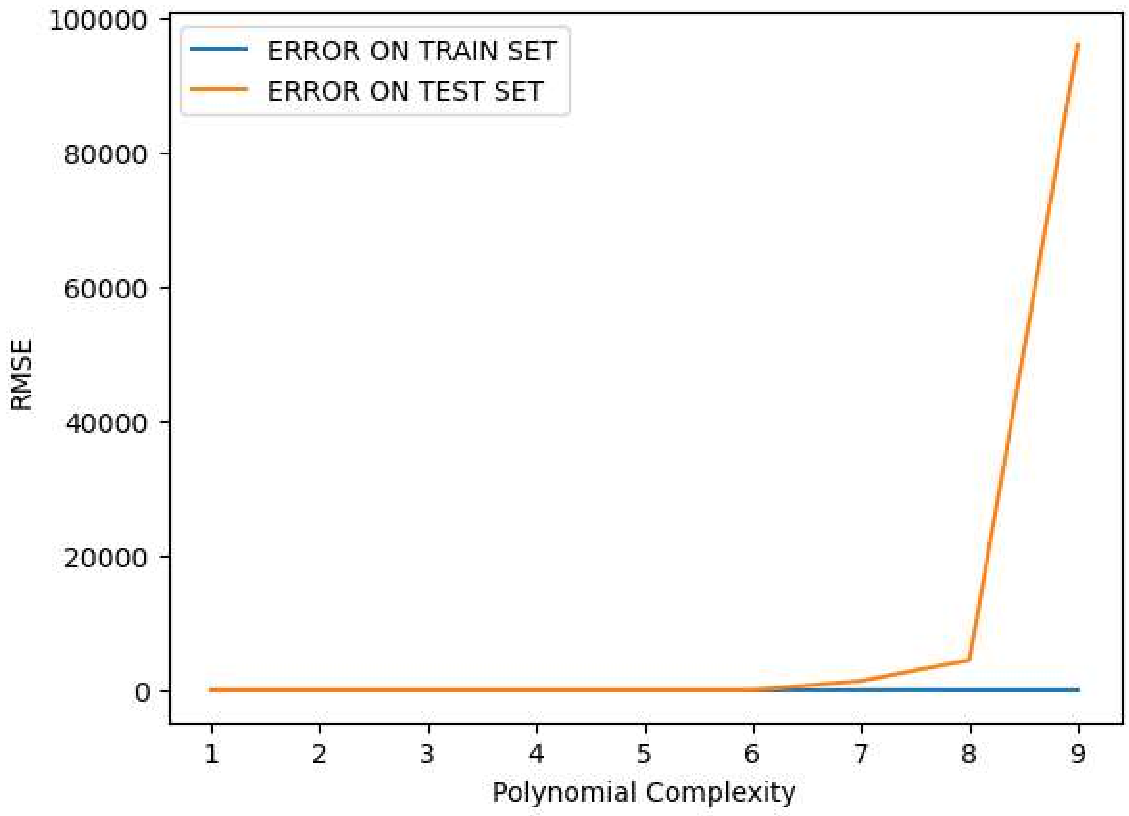
Another issue that we need to consider is that as we increase model complexity it clearly performs better. However, only on the training data set. Once we build a model that fits too many points in the training set (concept knowing as overfitting). The same model performs purely on unseen testing data. That’s why in order to figure out which polynomial order is good fit. We need to look not only to an error obtained during building the model. But, also check the error on the testing set.

Past experience has shown that it is very common as the polynomial complexity increases error on the training set keeps going down. However, at one point, after certain degree the error on the testing set will start going higher a lot.

```
# TRAINING ERROR PER DEGREE
train_rmse_errors = []
# TEST ERROR PER DEGREE
test_rmse_errors = []

for d in range(1,10):
     # CREATE POLY DATA SET FOR DEGREE "d"
    polynomial_converter = PolynomialFeatures(degree=d,include_bias=False)
    poly_features = polynomial_converter.fit_transform(X)

    # SPLIT THIS NEW POLY DATA SET
    X_train, X_test, y_train, y_test = train_test_split(poly_features, y, test_size=0.3,
random_state=101)

    # TRAIN ON THIS NEW POLY SET
    model = LinearRegression(fit_intercept=True)
    model.fit(X_train,y_train)

    # PREDICT ON BOTH TRAIN AND TEST
    train_pred = model.predict(X_train)
    test_pred = model.predict(X_test)

    # Calculate Errors
    # Errors on Train Set
    train_RMSE = np.sqrt(mean_squared_error(y_train,train_pred))
    # Errors on Test Set
    test_RMSE = np.sqrt(mean_squared_error(y_test,test_pred))
    train_rmse_errors.append(train_RMSE)
    test_rmse_errors.append(test_RMSE)
```

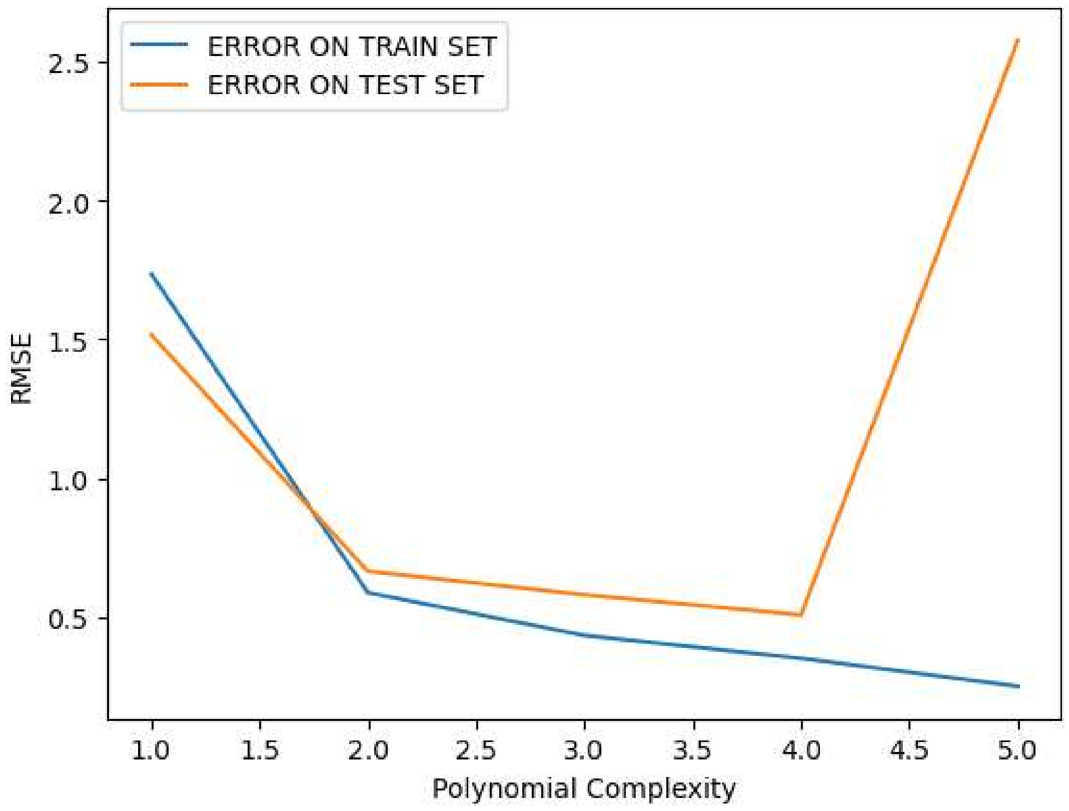
Plotting the results.

plt.plot(range(1,10),train_rmse_errors[:9],label='ERROR ON TRAIN SET')
plt.plot(range(1,10),test_rmse_errors[:9],label='ERROR ON TEST SET')
plt.xlabel("Polynomial Complexity")
plt.ylabel("RMSE")
plt.legend()



```
plt.plot(range(1,6),train_rmse_errors[:5],label='ERROR ON TRAIN SET')
plt.plot(range(1,6),test_rmse_errors[:5],label='ERROR ON TEST SET')
plt.xlabel("Polynomial Complexity")
plt.ylabel("RMSE")
plt.legend()
```

```
atplotlib.legend.Legend at 0x15c81a8ed10>
```



As we can seen after degree of 4 the error on the training sets begins to spike.

Which polynomial order is good fit. Even thought up to degree of 4 both error metrics keep going down, just to be on the safe side I would say degree =3 is a good choice.

Finalizing the model.

final_converter = PolynomialFeatures(degree=3,include_bias=False)
model_deploy = LinearRegression()
model_deploy.fit(final_converter.fit_transform(X),y)

If we are satisfied with model performance, then chances are we will use it in production. Hence we need to save the model and the converter and after that anybody can use it to predict unit sales given TV, Radio and newspaper ad spend.

Saving the model and polynomial converter.

from joblib import dump, load
dump(model_deploy, 'sales_poly_model.joblib')

dump(final_converter,'poly_converter.joblib')

Loading the model.

When somebody else wants to use the model they need to load it along with the polynomial converter.

loaded_poly_converter = load('poly_converter.joblib')
loaded_model = load('sales_poly_model.joblib')

Pick the first data available in the data set for simplicity.

adspend = [[230,37.8,69.2]]

adspend_poly = loaded_poly_converter.transform(adspend)

model_deploy.predict(adspend_poly)

array([21.24651338])

model_deploy.predict(adspend_poly)[0]

the predicted sale is 21.246513384209578.

If we check the original data set. The actual sales are 22.1.

error = (21.27-22.1)/22.1= 0.0376 * 100= 3.75 %

Not a bad performance.