import required library

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

loading data set

In [3]:
```
df=pd.read_csv('C:\\Users\\saran\\Downloads\\PythonForML_DS_Notes\\08-Linea
```

In [4]:
```
df.head()
```

Out[4]:

	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

In [5]:
```
df.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
```

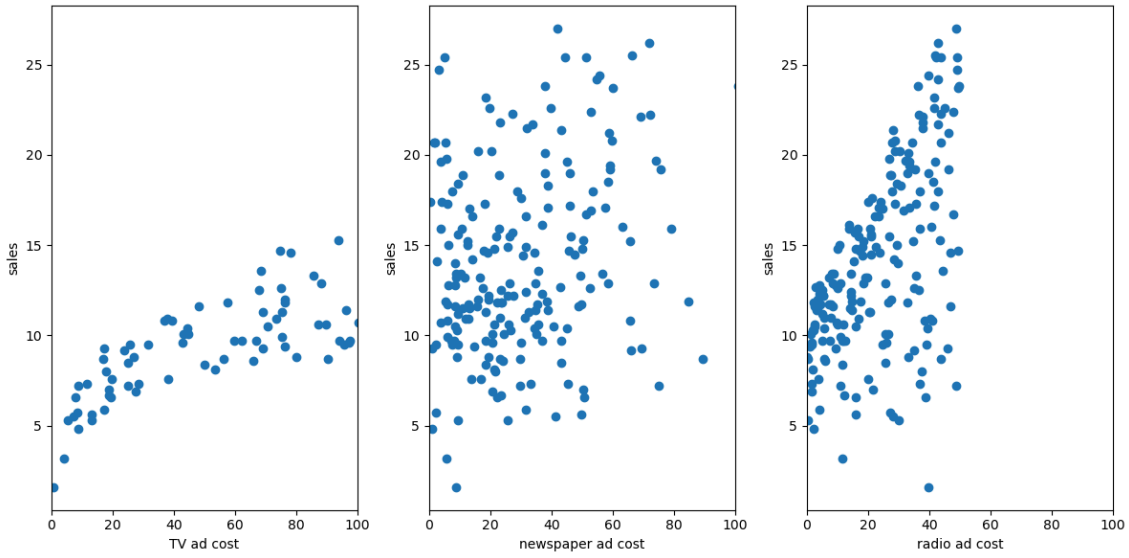
plotting relationship between different advertisement methods and sales

In [6]:
```
fig,axes=plt.subplots(nrows=1,ncols=3,figsize=(12,6))
#plotting the curve between sales and TV
axes[0].scatter(x='TV',y='sales',data=df)
axes[0].set_xlabel('TV ad cost')
axes[0].set_ylabel('sales')
axes[0].set_xlim((0,100))

#plotting the curve between sales and newspaper
axes[1].scatter(x='newspaper',y='sales',data=df)
axes[1].set_xlabel('newspaper ad cost')
axes[1].set_ylabel('sales')
axes[1].set_xlim((0,100))

#plotting the curve between sales and radio
axes[2].scatter(x='radio',y='sales',data=df)
axes[2].set_xlabel('radio ad cost')
axes[2].set_ylabel('sales')
axes[2].set_xlim((0,100))

plt.tight_layout()
```



Data separation into x-label and y-label

In [7]:
```
X=df.drop('sales',axis=1)
```

In [8]:
```
y=df['sales']
```

In [9]:
```
X.head()
```

Out[9]:

	TV	radio	newspaper
0	230.1	37.8	69.2
1	44.5	39.3	45.1
2	17.2	45.9	69.3
3	151.5	41.3	58.5
4	180.8	10.8	58.4

In [10]:
```
y.head()
```

Out[10]:
```
0    22.1
1    10.4
2     9.3
3    18.5
4    12.9
Name: sales, dtype: float64
```

splitting data into training and test set

In [11]:
```
from sklearn.model_selection import train_test_split
```

In [12]:
```
 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
```

70% data is training data and 30% data is test data

A random_state(49) is chosen so splitting of data will be same everytime

In [13]:
```
X_train.head()
```

Out[13]:

	TV	radio	newspaper
15	195.4	47.7	52.9
93	250.9	36.5	72.3
185	205.0	45.1	19.6
157	149.8	1.3	24.3
69	216.8	43.9	27.2

In [14]:
```
X_test.head()
```

Out[14]:

	TV	radio	newspaper
107	90.4	0.3	23.2
144	96.2	14.8	38.9
104	238.2	34.3	5.3
195	38.2	3.7	13.8
153	171.3	39.7	37.7

feature scaling

In [15]:
```
from sklearn.preprocessing import StandardScaler
```

In [16]:
```
scaler=StandardScaler()
```

In [17]:
```
scaler.fit(X_train)
```

Out[17]:
```
StandardScaler()
```

In [18]:
```
X_train=scaler.transform(X_train)
```

In [19]:
```
X_train[:5]
```

Out[19]:
```
array([[ 5.29893869e-01,  1.62386791e+00,  9.69990993e-01],
       [ 1.17467899e+00,  8.44156714e-01,  1.86171031e+00],
       [ 6.41424267e-01,  1.44286353e+00, -5.60640611e-01],
       [ 1.24475891e-04, -1.60636419e+00, -3.44605519e-01],
       [ 7.78513716e-01,  1.35932304e+00, -2.11307272e-01]])
```

In [20]:
```
X_test=scaler.transform(X_test)
```

In [21]:
```
X_test[:5]
```

Out[21]:
```
array([[-0.68996987, -1.67598126, -0.39516692],
       [-0.62258692, -0.66653373,  0.32648221],
       [ 1.02713356,  0.69099916, -1.21793887],
       [-1.29641641, -1.43928322, -0.82723711],
       [ 0.2499061 ,  1.06693134,  0.27132432]])
```

Linear regression without regularised terms

In [22]:
```
from sklearn.linear_model import LinearRegression
```

In [23]:
```
model1=LinearRegression()
```

fitting training data into linear regression model

In [24]:
```
model1.fit(X_train,y_train)
```

Out[24]:
```
LinearRegression()
```

predicting train and test error

In [25]:
```
y_train_pred_model1=model1.predict(X_train)
```

In [26]:
```
y_test_pred_model1=model1.predict(X_test)
```

finding mean absolute error, mean squared error and root mean squared error for training and test set

In [27]:
```
from sklearn.metrics import mean_absolute_error,mean_squared_error
```

In [28]:
```
MAE_model1_train_error=mean_absolute_error(y_train,y_train_pred_model1)
MSE_model1_train_error=mean_squared_error(y_train,y_train_pred_model1)
RMSE_model1_train_error=np.sqrt(MSE_model1_train_error)
```

In [29]:
```
MAE_model1_test_error=mean_absolute_error(y_test,y_test_pred_model1)
MSE_model1_test_error=mean_squared_error(y_test,y_test_pred_model1)
RMSE_model1_test_error=np.sqrt(MSE_model1_test_error)
```

error values in train and test data set using error metrics

In [30]:
```
MAE_model1_train_error
```

Out[30]:
```
1.1531222562529329
```

In [31]:
```
MAE_model1_test_error
```

Out[31]:
```
1.487703814408449
```

In [32]:
```
MSE_model1_train_error
```

Out[32]:
```
2.296461469424213
```

In [33]:
```
MSE_model1_test_error
```

Out[33]:
```
4.031955199416205
```

In [34]:
```
RMSE_model1_train_error
```

Out[34]:
```
1.5154080207733538
```

In [35]:
```
RMSE_model1_test_error
```

Out[35]:
```
2.007972908038404
```

creating final model using complete data set

In [36]:
```
final_model1=LinearRegression()
```

In [37]:
```
final_model1.fit(X,y)
```

Out[37]:
```
LinearRegression()
```

polynomial regression

In [38]:
```
from sklearn.preprocessing import PolynomialFeatures
```

In polynomial regression exisiting model is made complex by introducing higher degree terms

choosing correct degree of model is important

In [39]:
```
train_error_model2=[]
test_error_model2=[]
for d in range(1,10):
    #creating instance of PolynomialFeatures
    polynomial_converter=PolynomialFeatures(degree=d,include_bias=False)

    #creating polynomial features for training and test set
    poly_features_train=polynomial_converter.fit_transform(X_train)
    poly_features_test=polynomial_converter.transform(X_test)

    #creating instance of LinearRegression()model
    model2=LinearRegression()

    #fitting training data(tuning parameters)
    model2.fit(poly_features_train,y_train)

    #predicting test and train value
    y_train_pred_model2=model2.predict(poly_features_train)
    y_test_pred_model2=model2.predict(poly_features_test)

    #calculating and storing train and test error
    train_error_model2.append(mean_squared_error(y_train,y_train_pred_model
    test_error_model2.append(mean_squared_error(y_test,y_test_pred_model2))
```

In [40]:
```
train_error_model2
```

Out[40]:
```
[2.296461469424213,
 0.24504471520011148,
 0.1321292424663568,
 0.08133875357271667,
 0.048684503291186725,
 0.035195733800655195,
 0.014853287325468572,
 1.1716942655433484e-22,
 4.715907758749683e-22]
```

In [41]:
```
test_error_model2
```

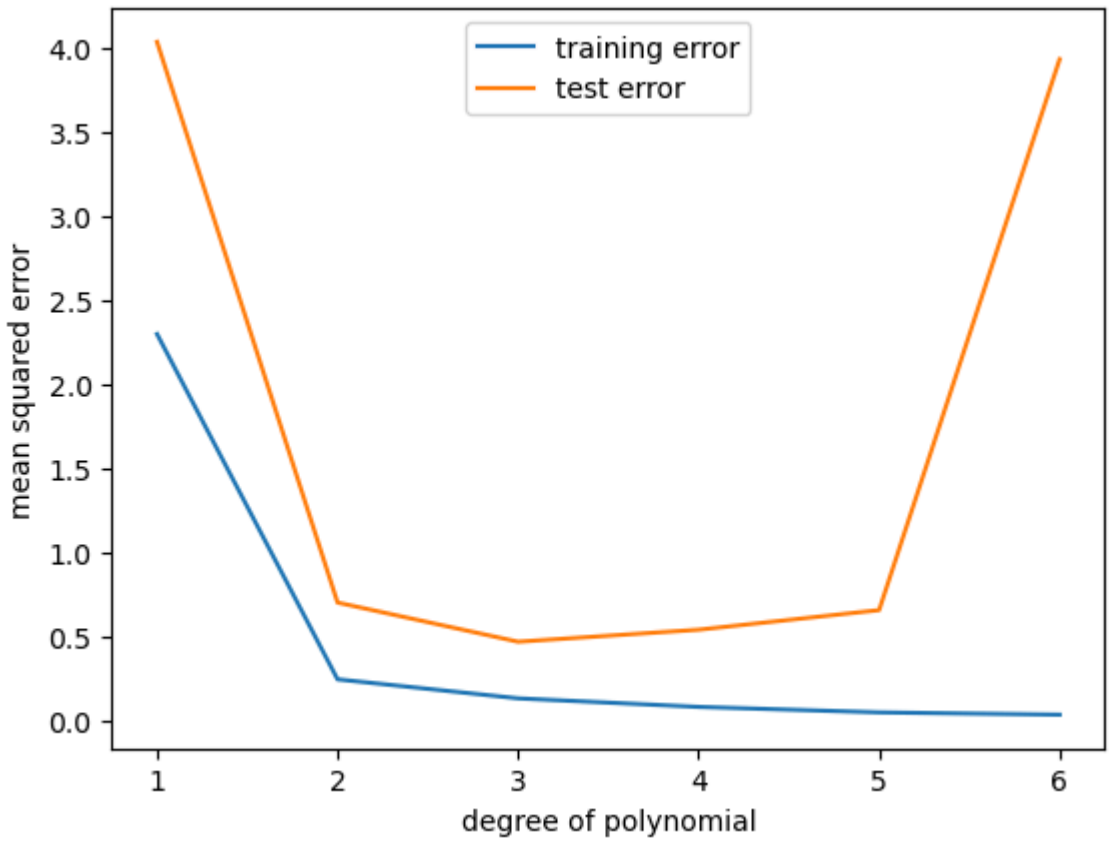
Out[41]:
```
[4.031955199416205,
 0.7011728329627888,
 0.46844681722448783,
 0.5398708574582972,
 0.6562875157223154,
 3.92901028522235,
 193.67817883380175,
 53781.71202944154,
 46435.083754490595]
```

In [42]:
```
degree=range(1,7)
plt.plot(degree,train_error_model2[:6],label='training error')
plt.plot(degree,test_error_model2[:6],label='test error')

plt.xlabel('degree of polynomial')
plt.ylabel('mean squared error')

plt.legend()
```

Out[42]:
```
<matplotlib.legend.Legend at 0x21dba3229a0>
```



clearly model of degree 3 has lowest test error also training error is low

the trade-off between train-test error and complexity of model i.e. degree is best for the model of degree 3

creating polynomial model

In [43]:
```
polynomial_converter=PolynomialFeatures(degree=3,include_bias=False)
```

In [44]:
```
polynomial_converter.fit(X_train)
X_train_poly_feature=polynomial_converter.transform(X_train)
X_test_poly_feature=polynomial_converter.transform(X_test)
```

In [45]:
```
model2=LinearRegression()
```

In [46]:
```
model2.fit(X_train_poly_feature,y_train)
```

Out[46]:
```
LinearRegression()
```

In [47]:
```
y_train_pred_model2=model2.predict(X_train_poly_feature)
y_test_pred_model2=model2.predict(X_test_poly_feature)
```

In [48]:
```
MAE_model2_train_error=mean_absolute_error(y_train,y_train_pred_model2)
MSE_model2_train_error=mean_squared_error(y_train,y_train_pred_model2)
RMSE_model2_train_error=np.sqrt(MSE_model2_train_error)
```

In [49]:
```
MAE_model2_test_error=mean_absolute_error(y_test,y_test_pred_model2)
MSE_model2_test_error=mean_squared_error(y_test,y_test_pred_model2)
RMSE_model2_test_error=np.sqrt(MSE_model2_test_error)
```

computing train and test error

In [50]:
```
MAE_model2_train_error
```

Out[50]:
```
0.27290440284296336
```

In [51]:
```
MSE_model2_train_error
```

Out[51]:
```
0.1321292424663568
```

In [52]:
```
RMSE_model2_train_error
```

Out[52]:
```
0.36349586306635845
```

In [53]:
```
MAE_model2_test_error
```

Out[53]:
```
0.4065722444776206
```

In [54]:
```
MSE_model2_test_error
```

Out[54]:
```
0.46844681722448783
```

In [55]:
```
RMSE_model2_test_error
```

Out[55]:
```
0.6844317476742936
```

creating final model using complete data set

In [56]:
```
final_polynomial_converter=PolynomialFeatures(degree=3,include_bias=False)
```

In [57]:
```
final_poly_feature=final_polynomial_converter.fit_transform(X)
```

In [58]:
```
final_model2=LinearRegression()
```

In [59]:
```
final_model2.fit(final_poly_feature,y)
```

Out[59]:
```
LinearRegression()
```

Linear regression using regularised term (L1 regularisation:Lasso regression)

In [60]:
```
#importing Lasso model with cross validation set
from sklearn.linear_model import LassoCV
```

In [61]:
```
#creating instance of LassoCV
model3=LassoCV(eps=0.1,n_alphas=100,max_iter=1000)
```

In [62]:
```
#fitting training data(tuning parameters)
model3.fit(X_train_poly_feature,y_train)
```

Out[62]:
```
LassoCV(eps=0.1)
```

chosen value of alpha i.e. lambda(regularisation term)

In [63]:
```
model3.alpha_
```

Out[63]:
```
0.7863850361607775
```

In [64]:
```
#predicting train and test values
y_train_pred_model3=model3.predict(X_train_poly_feature)
y_test_pred_model3=model3.predict(X_test_poly_feature)
```

In [65]:
```
#predicting train error from different error metrics
MAE_model3_train_error=mean_absolute_error(y_train,y_train_pred_model3)
MSE_model3_train_error=mean_squared_error(y_train,y_train_pred_model3)
RMSE_model3_train_error=np.sqrt(MSE_model3_train_error)
```

In [66]:
```
#predicting test error from different error metrics
MAE_model3_test_error=mean_absolute_error(y_test,y_test_pred_model3)
MSE_model3_test_error=mean_squared_error(y_test,y_test_pred_model3)
RMSE_model3_test_error=np.sqrt(MSE_model3_test_error)
```

computing train and test error

In [67]:
```
MAE_model3_train_error
```

Out[67]:
```
1.5177033182952147
```

In [68]:
```
MAE_model3_test_error
```

Out[68]:
```
1.6364791937803005
```

In [69]:
```
MSE_model3_train_error
```

Out[69]:
```
3.6052068394435586
```

In [70]:
```
MSE_model3_test_error
```

Out[70]:
```
4.315683871937832
```

In [71]:
```
RMSE_model3_train_error
```

Out[71]:
```
1.8987382229900884
```

In [72]:
```
RMSE_model3_test_error
```

Out[72]:
```
2.0774224105698464
```

In Lasso regression many parameters are assigned with 0 value which decreases the complexity of model

In [73]:
```
model3.coef_
```

Out[73]:
```
array([ 0.56340285,  0.        ,  0.        , -0.        ,  0.31959244,
        0.        , -0.        , -0.        ,  0.        ,  1.51286642,
        0.        ,  0.        ,  0.        ,  0.        ,  0.05911565,
        1.16701337,  0.        ,  0.        ,  0.        ])
```

Linear Regression using regularised term L2(Ridge regression)

In [74]:
```
#importing ridge model with cross validation set
from sklearn.linear_model import RidgeCV
```

In [75]:
```
#creating instance of RidgeCV()
model4=RidgeCV(cv=10)
```

In [76]:
```
#fitting model with training data(tuning parameters)
model4.fit(X_train_poly_feature,y_train)
```

Out[76]:
```
RidgeCV(alphas=array([ 0.1,  1. , 10. ]), cv=10)
```

In [77]:
```
#predicting training and test value
y_train_pred_model4=model4.predict(X_train_poly_feature)
y_test_pred_model4=model4.predict(X_test_poly_feature)
```

In [78]:
```
#predicting training errors
MAE_model4_train_error=mean_absolute_error(y_train,y_train_pred_model4)
MSE_model4_train_error=mean_squared_error(y_train,y_train_pred_model4)
RMSE_model4_train_error=np.sqrt(MSE_model4_train_error)
```

In [79]:
```
#predicting test errors
MAE_model4_test_error=mean_absolute_error(y_test,y_test_pred_model4)
MSE_model4_test_error=mean_squared_error(y_test,y_test_pred_model4)
RMSE_model4_test_error=np.sqrt(MSE_model4_test_error)
```

computing mean absolute error

In [80]:
```
MAE_model4_train_error
```

Out[80]:
```
0.27316731082699075
```

In [81]:
```
MAE_model4_test_error
```

Out[81]:
```
0.40771917251614287
```

In [82]:
```
MSE_model4_train_error
```

Out[82]:
```
0.13220759302330634
```

In [83]:
```
MSE_model4_test_error
```

Out[83]:
```
0.4693862382195732
```

In [84]:
```
RMSE_model4_train_error
```

Out[84]:
```
0.3636036207510953
```

In [85]:
```
RMSE_model4_test_error
```

Out[85]:
```
0.6851176820222736
```

Which model is best

In [86]:
```
df['sales'].mean()
```

Out[86]:
```
14.0225
```

since the mean is 14.0225 the mean absolute error upto 1.40225 i.e. 10% of mean value is acceptable in this case

polynomial regression has minimum mean absolute error and root mean squared error i.e. 0.4065722444776206 and 0.6844317476742936 respectively

polynomial regression has mean absolute error 0.40 close to 3% of mean value which is exceptional

Therefore polynomial regression will do best prediction

on the other hand model4 i.e. RidgeCV regression is more robust and simpler model also mean absolute error is 0.40771917251614287 and root mean squared error is 0.6851176820222736 which is quite close to mean absolute error and root mean squared error of polynomial regression

model selection based on robustness and best output prediction

1) RidgeCV Regression

2) Polynomial Regression

3) Linear Regression

4) LassoCV Regression

since RidgeCV is less complex model and error is nearly equal to errors in polynomial regression therefore RidgeCV model

is best model for this data set

Model Deployment

choosing final polynomial converter

In [93]:
```
final_poly_converter = PolynomialFeatures(degree=3,include_bias=False)
```

In [91]:
```
#importing joblib
from joblib import dump
```

In [89]:
```
dump(model4,'AdvertisementModel.joblib')
```

Out[89]:
```
['AdvertisementModel.joblib']
```

In [94]:
```
dump(final_poly_converter,'final_polynomial_converter.joblib')
```

Out[94]:
```
['final_polynomial_converter.joblib']
```

This model can be used further by loading the model using joblib.load('AdvertisementModel.joblib') and joblib.load('final_polynomial_converter')