Advertising Dataset

This sample data displays sales (in thousands of units) for a particular product as a function of advertising budgets (in thousands of dollars) for TV, radio, and newspaper media.

Indepenent variables

- TV: Advertising dollars spent on TV for a single product in a given market (in thousands of dollars)
- Radio: Advertising dollars spent on Radio
- Newspaper: Advertising dollars spent on Newspaper

Target Variable

- Sales: sales of a single product in a given market (in thousands of widgets)

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

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
```

In [2]:
```
# read data into a DataFrame
df = pd.read_csv("Advertising.csv")
df.head()
```

Out[2]:

	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 [3]:
```
# print the shape of the DataFrame
df.shape
```

Out[3]:
```
(200, 4)
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['TV', 'radio', 'newspaper', 'sales'], dtype='object')
```

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

Out[5]:
```
TV           float64
radio        float64
newspaper    float64
sales        float64
dtype: object
```

In [6]:
```
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
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
TV           0
radio        0
newspaper    0
sales        0
dtype: int64
```

Question :

Our next ad campaign will have a total spend of $200, how many units do we expect to sell as a result of this?

If someone was to spend a total of $200 , what would the expected sales be? We have simplified this quite a bit by combining all the features into "total spend", but we will come back to individual features later on. For now, let's focus on understanding what a linear regression line can help answer.

Simple Linear Regression

In [8]:
```
df['total_spend'] = df['TV'] + df['radio'] + df['newspaper']
df.head()
```

Out[8]:

	TV	radio	newspaper	sales	total_spend
0	230.1	37.8	69.2	22.1	337.1
1	44.5	39.3	45.1	10.4	128.9
2	17.2	45.9	69.3	9.3	132.4
3	151.5	41.3	58.5	18.5	251.3
4	180.8	10.8	58.4	12.9	250.0

In [9]:
```
df.drop(columns=['TV','radio','newspaper'],inplace=True)
df
```

Out[9]:

	sales	total_spend
0	22.1	337.1
1	10.4	128.9
2	9.3	132.4
3	18.5	251.3
4	12.9	250.0
...	...	...
195	7.6	55.7
196	9.7	107.2
197	12.8	192.7
198	25.5	391.8
199	13.4	249.4

200 rows × 2 columns

EDA

On the basis of this data, how should you spend advertising money in the future? These general questions might lead you to more specific questions:

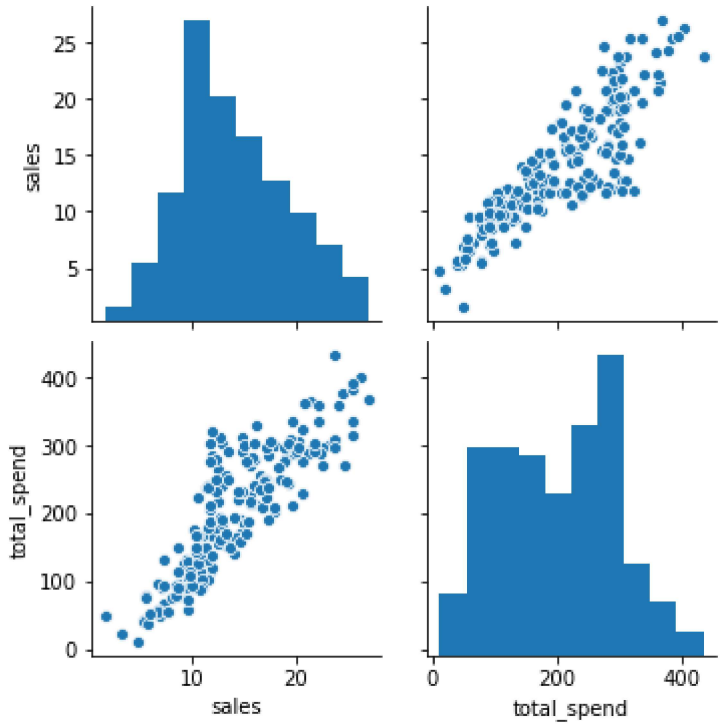
1. Is there a relationship between ads and sales?
2. How strong is that relationship?
3. Given ad spending, can sales be predicted?

In [10]:
```
df.describe()
```

Out[10]:

	sales	total_spend
count	200.000000	200.000000
mean	14.022500	200.860500
std	5.217457	92.985181
min	1.600000	11.700000
25%	10.375000	123.550000
50%	12.900000	207.350000
75%	17.400000	281.125000
max	27.000000	433.600000

In [11]:
```
sns.pairplot(df)
plt.show()
```



In [12]:
```
df.corr()
```

Out[12]:

	sales	total_spend
sales	1.000000	0.867712
total_spend	0.867712	1.000000

In [13]:
```
# create X and y
# taking only one variable for now

X = df[['total_spend']]
y = df['sales']
```

Train | Test Split

Make sure you have watched the Machine Learning Overview videos on Supervised Learning to understand why we do this step

In [14]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,random
_state=42)
```

Least Squares Line

Simply solve for m and b, remember we are solving in a generalized form:

$$\hat{y} = \beta_0 + \beta_1 X$$

Capitalized to signal that we are dealing with a matrix of values, we have a known matrix of labels (sales numbers) Y and a known matrix of total_spend (X). We are going to solve for the *beta* coefficients, which as we expand to more than just a single feature, will be important to build an understanding of what features have the most predictive power. We use y hat to indicate that y hat is a prediction or estimation, y would be a true label/known value.

Modelling

Creating a Model (Estimator)

In [15]:
```
# follow the usual sklearn pattern: import, instantiate, fit

from sklearn.linear_model import LinearRegression #import

model = LinearRegression()   #instantiate

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

Out[15]:
```
LinearRegression()
```

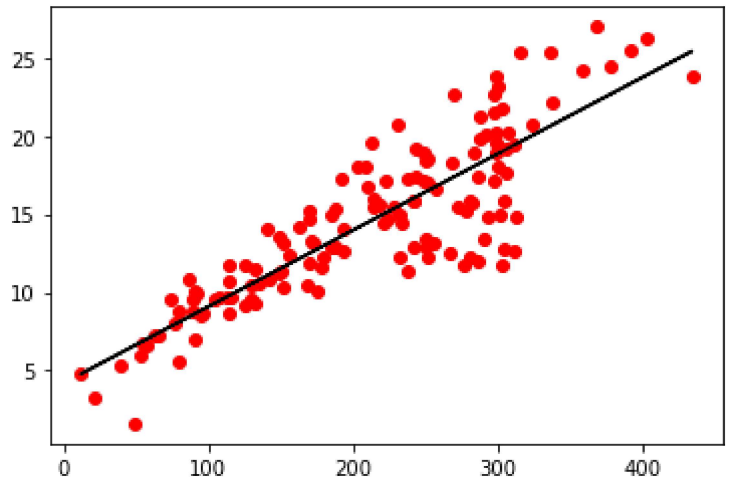
In [16]:
```
# print intercept and coefficients
model.intercept_ , model.coef_
```

Out[16]:
```
(4.199106500868718, array([0.04895385]))
```

In [17]:
```
train_predictions = model.predict(X_train)
test_predictions = model.predict(X_test)
```

Plotting the Least Squares Line

In [18]:
```
plt.scatter(X_train,y_train,color='red')
plt.plot(X_train,train_predictions,color='black')
plt.show()
```



Evaluation Metrics

Make sure you've viewed the video on these metrics! The three most common evaluation metrics for regression problems:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Comparing these metrics:

- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are **loss functions**, because we want to minimize them.

In [19]:
```
train_res = y_train - train_predictions
test_res = y_test - test_predictions
```

In [20]:
```
model.score(X_train,y_train)  #R-Squared value (train)
```

Out[20]: 0.7404192834391546

In [21]:
```
model.score(X_test,y_test)   #R-Squared value (test)
```

Out[21]: 0.7650262463310722

In [22]:
```
from sklearn.metrics import r2_score
r2_score(y_test,test_predictions)
```

Out[22]: 0.7650262463310722

In [23]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model,X,y,cv=5)
scores

# Average of the MSE scores (we set back to positive)
abs(scores.mean())
```

Out[23]:
```
0.743378317855542
```

In [24]:
```
# from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

In [25]:
```
from sklearn.metrics import mean_absolute_error
MAE = mean_absolute_error(y_test,test_predictions)
MAE
```

Out[25]:
```
1.9143627368130551
```

In [26]:
```
from sklearn.metrics import mean_squared_error
MSE = mean_squared_error(y_test,test_predictions)
MSE
```

Out[26]:
```
6.41586631254322
```

In [27]:
```
RMSE = np.sqrt(MSE)
RMSE
```

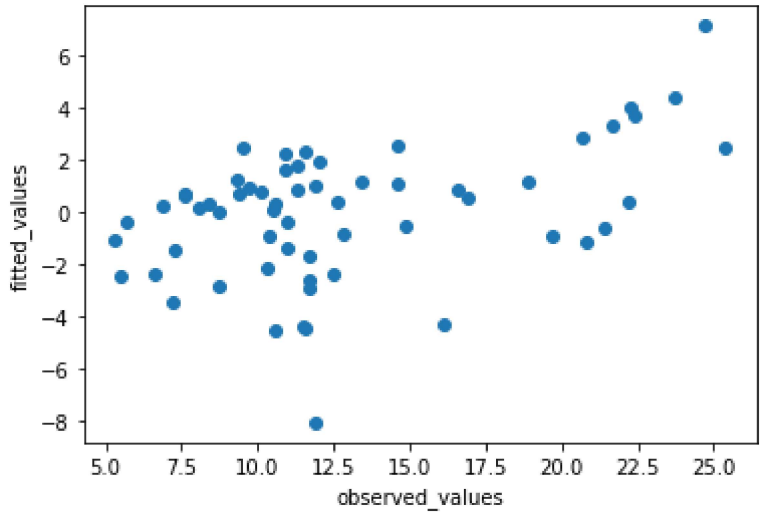
Out[27]:
```
2.5329560423629975
```

Diagnosis Test

It's also important to plot out residuals, this helps us understand if Linear Regression was a valid model choice.

1. Linearity (Observed values VS Fitted values)

In [28]:
```
plt.scatter(y_test,test_res)
plt.xlabel("observed_values")
plt.ylabel("fitted_values")
plt.show()
```

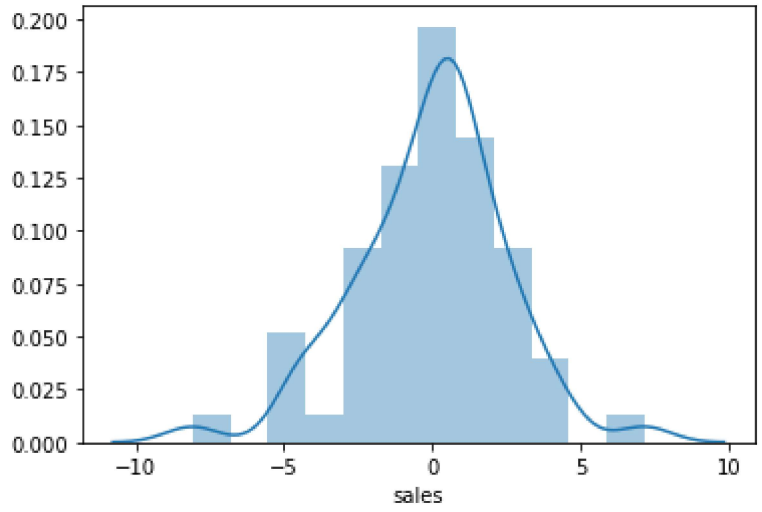


2. Normality of Residuals

It's also important to plot out residuals and check for normal distribution

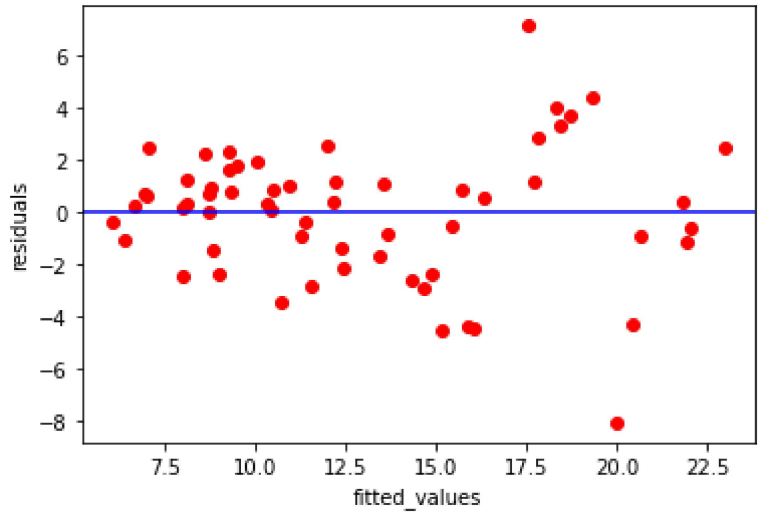
In [29]:
```
sns.distplot(test_res,kde=True)

#plt.hist(test_res)
#plt.show()
```

Out[29]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x19cce5b6670>
```



3. Homoscadesicity (y_hat vs residuals)

In [30]:
```
# Homoscedasticity (Residuals VS Fitted Values)
plt.scatter(test_predictions,test_res,c="r")
plt.axhline(y=0,color='blue')
plt.xlabel("fitted_values")
plt.ylabel("residuals")
plt.show()
```



In [31]:
```
spend = 200
predicted_sales = 4.199 + 0.04895*spend
predicted_sales
```

Out[31]: 13.989

In [32]:
```
X_new = pd.DataFrame({'total_spend': [200]})
X_new
```

Out[32]:

	total_spend
0	200

In [33]:
```
# use the model to make predictions on a new value
model.predict(X_new)
```

Out[33]: array([13.98987609])

Confidence in the Model

Question: Is linear regression a high bias/low variance model, or a low bias/high variance model?

Answer: It's a High bias/low variance model. Under repeated sampling, the line will stay roughly in the same place (low variance), but the average of those models won't do a great job capturing the true relationship (high bias). Note that low variance is a useful characteristic when you don't have a lot of training data.

A closely related concept is **confidence intervals**. Statsmodels calculate 95% confidence intervals for your model coefficients, which are interpreted as follows: If the population from which this sample was drawn was **sampled 100 times**, approximately **95 of those confidence intervals** would contain the "true" coefficient.

In [34]:
```
#importing required libraries
import statsmodels.formula.api as smf

#model
model1=smf.ols("y~X",data=df).fit()

# P-values for the variables and R-squared value for prepared model
model1.summary()
```

Out[34]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.753
Model:	OLS	**Adj. R-squared:**	0.752
Method:	Least Squares	**F-statistic:**	603.4
Date:	Mon, 20 Sep 2021	**Prob (F-statistic):**	5.06e-62
Time:	21:05:48	**Log-Likelihood:**	-473.88
No. Observations:	200	**AIC:**	951.8
Df Residuals:	198	**BIC:**	958.4
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	4.2430	0.439	9.676	0.000	3.378	5.108
X	0.0487	0.002	24.564	0.000	0.045	0.053

Omnibus:	6.851	**Durbin-Watson:**	1.967
Prob(Omnibus):	0.033	**Jarque-Bera (JB):**	6.692
Skew:	-0.373	**Prob(JB):**	0.0352
Kurtosis:	3.495	**Cond. No.**	528.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Hypothesis Testing and p-values

Closely related to confidence intervals is **hypothesis testing**. Generally speaking, you start with a **null hypothesis** and an **alternative hypothesis** (that is opposite the null). Then, you check whether the data supports **rejecting the null hypothesis** or **failing to reject the null hypothesis**.

(Note that "failing to reject" the null is not the same as "accepting" the null hypothesis. The alternative hypothesis may indeed be true, except that you just don't have enough data to show that.)

As it relates to model coefficients, here is the conventional hypothesis test:

- **null hypothesis:** There is no relationship between totalspend and Sales (and thus β_1 equals zero)
- **alternative hypothesis:** There is a relationship between totalspend and Sales (and thus β_1 is not equal to zero)

How to test this hypothesis? Intuitively, reject the null (and thus believe the alternative) if the 95% confidence interval **does not include zero**. Conversely, the **p-value** represents the probability that the coefficient is actually zero:

Retraining Model on Full Data

If we're satisfied with the performance on the test data, before deploying our model to the real world, we should retrain on all our data. (If we were not satisfied, we could update parameters or choose another model, something we'll discuss later on).

In [35]:
```
final_model = LinearRegression()

final_model.fit(X,y)

final_model.coef_
```

Out[35]:
```
array([0.04868788])
```

Interpreting the coefficients:

- a 1 unit (A thousand dollars) increase in total_spend is associated with an increase in sales of 0.0486 "sales units", in this case 1000s of units .
- This basically means that for every $1000 dollars spend on total spend on Ads, we could expect 48 more units sold.

Prediction on New Data

Recall , X_test data set looks *exactly* the same as brand new data, so we simply need to call .predict() just as before to predict sales for a new advertising campaign.

Our next ad campaign will have a total spend of 200k on Ads, how many units could we expect to sell as a result of this?

```
In [36]: final_model.predict([[200]])
Out[36]: array([13.98060408])
```

How accurate is this prediction? No real way to know! We only know truly know our model's performance on the test data, that is why we had to be satisfied by it first, before training our full model

Deployment

Model Persistence (Saving and Loading a Model)

```
In [37]: from joblib import dump
In [38]: dump(final_model, 'sales_model.joblib')
Out[38]: ['sales_model.joblib']
In [39]: from joblib import load
In [40]: loaded_model = load('sales_model.joblib')
In [41]: loaded_model.intercept_
Out[41]: 4.243028216036329
In [42]: loaded_model.coef_
Out[42]: array([0.04868788])
In [43]: loaded_model.predict([[200]])
Out[43]: array([13.98060408])
```