LESSON 12 Correlation and CrossValidation

Correlation

Correlation is a score measure of the extent to which two variables are related.

Correlation also means association and is statistical summary of the relationship between variables.

For example:

- One variable could cause or depend on the values of another variable.
- One variable could be lightly associated with another variable.
- Two variables could depend on a third unknown variable.
- The performance of some algorithms can deteriorate if two or more variables are tightly related, called **multicollinearity**. An example is linear regression, where one of the offending correlated variables should be removed in order to improve the accuracy of the model.
- We may also be interested in the correlation between input variables with the output variable in order provide insight into which variables may or may not be relevant as input for developing a model.
- The structure of the relationship may be known, e.g. it may be linear, or we may have no idea whether a relationship exists between two variables or what structure it may take. Depending what is known about the relationship and the distribution of the variables, different correlation scores can be calculated.

Statistics and data science are often concerned about the relationships between two or more variables (or features) of a dataset. Each data point in the dataset is an **observation**, and the **features** are the properties or attributes of those observations.

Every dataset you work with uses variables and observations. For example, you might be interested in understanding the following:

- How the height of basketball players is correlated to their shooting accuracy

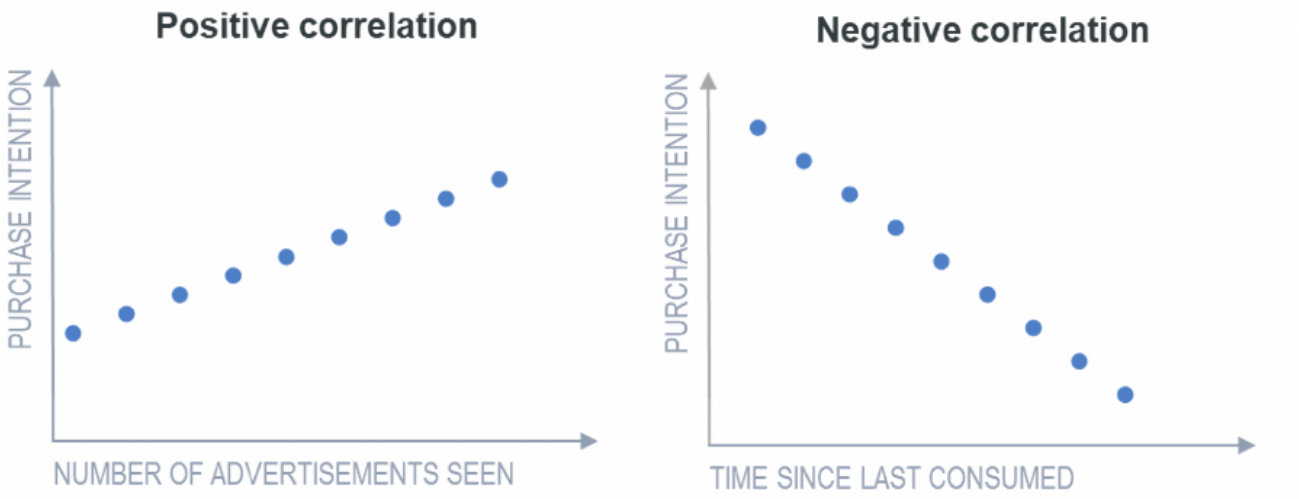
- Whether there's a relationship between employee work experience and salary
- What mathematical dependence exists between the population density and the gross domestic product of different countries

In the examples above, the height, shooting accuracy, years of experience, salary, population density, and gross domestic product are the **features** or **variables**. The data related to each player, employee, and each country are the **observations**.

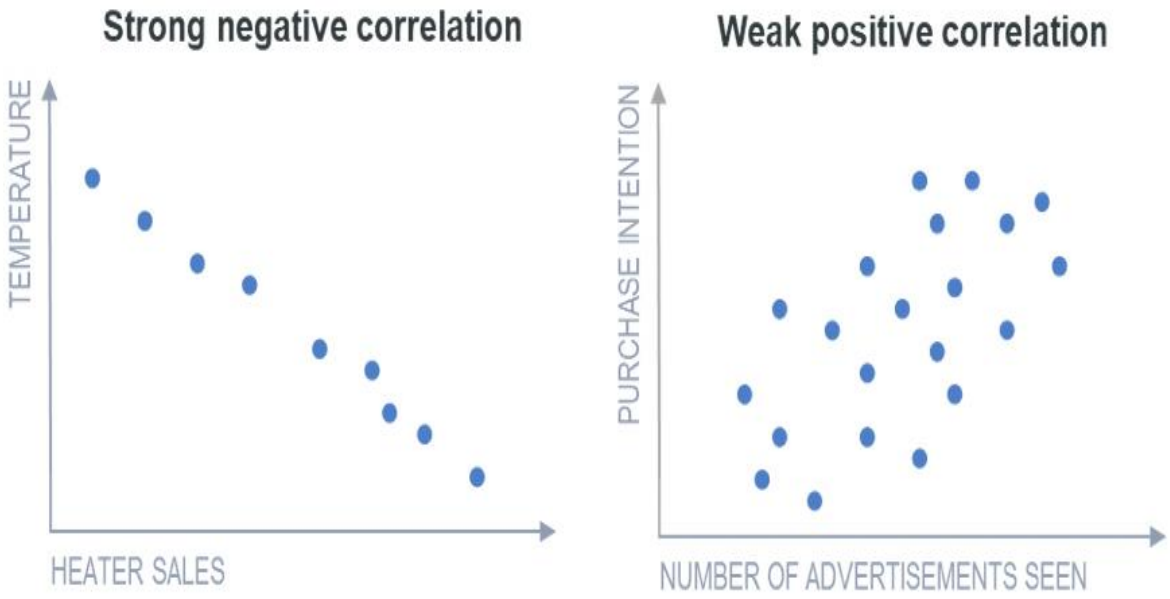
Positive and Negative Correlation

Positive correlation is a relationship between two variables in which both variables move in the same direction. This is when one variable increases while the other increases and vice versa. For example, positive correlation may be that the more you exercise, the more calories you will burn.

Negative correlation is a relationship where one variable increases as the other decreases, and vice versa.



When it is possible to predict, with a reasonably high level of accuracy, the values of one variable based on the values of the other, the relationship between the two variables is described as a **strong correlation**. A **weak correlation** is where on average the values of one variable are related to the other.

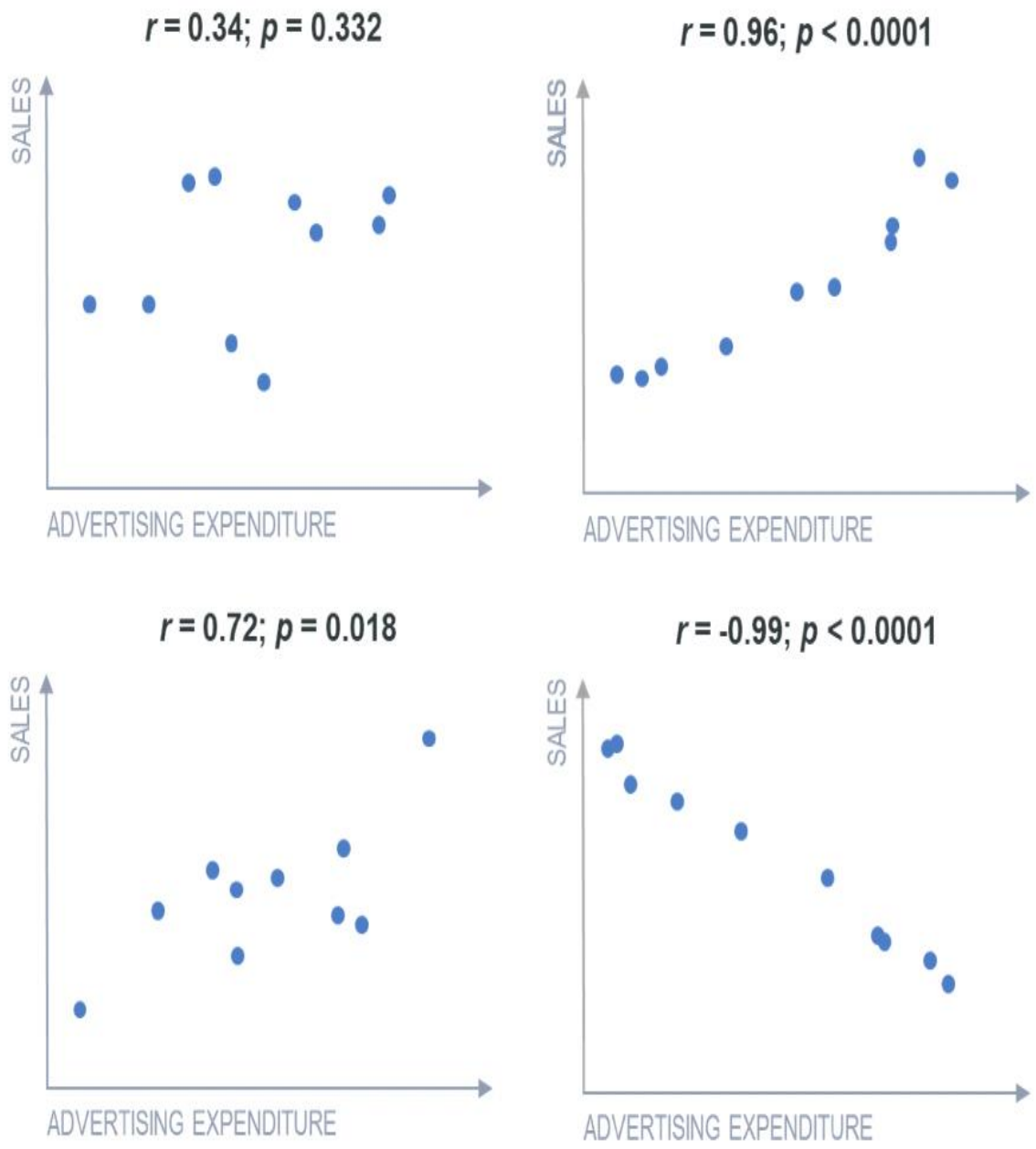


Pearson's Product-Moment Correlation

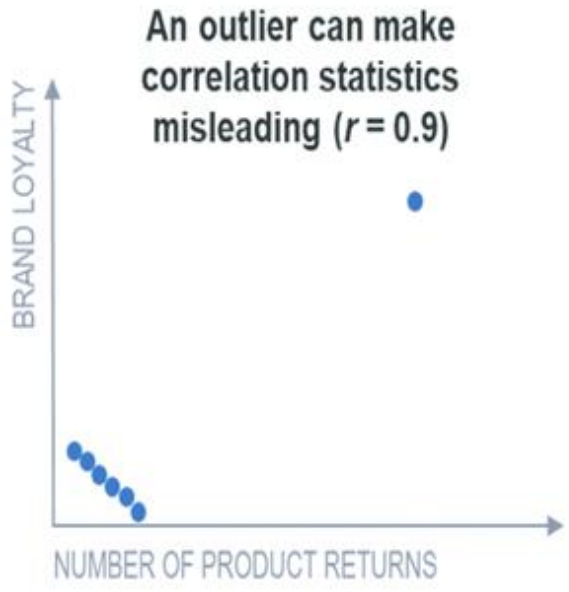
The most common measure of *correlation is* **Pearson's product-moment correlation,** which is commonly referred to simply as the correlation, the **correlation coefficient**, or just the letter ***r*** . The **correlation coefficient r** measures the strength and direction of a linear relationship, for instance:

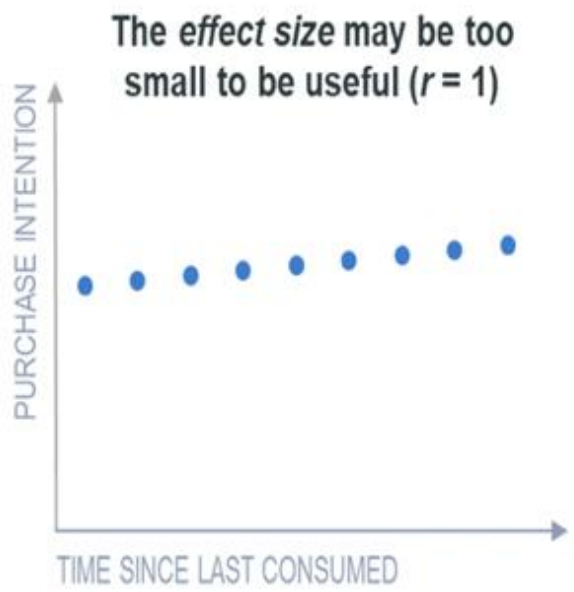
- 1 indicates a perfect positive correlation.
- -1 indicates a perfect negative correlation.
- 0 indicates that there is no relationship between the different variables.

Values between -1 and 1 denote the strength of the correlation, as shown in the example below.



Outliners re data points outside the normal data points, and result in inaccurate correlation r values. Closely related data sizes may also not give useful correlation results.





Another problem with *correlation* is that it summarizes a linear relationship. If the true relationship is nonlinear, then this may be missed. One more problem is that very high correlations often reflect tautologies (truths) rather than findings of interest.

Linear Correlation

Linear correlation measures the proximity of the mathematical relationship between variables or dataset features to a linear function. If the relationship between the two features is closer to some linear function, then their linear correlation is stronger and the absolute value of the correlation coefficient is higher.

Calculating Pearson Correlation Coefficient

Consider a dataset with two features: **x** and **y**. Each feature has n values, so **x** and **y** are n-tuples. Say that the first value x_1 from **x** corresponds to the first value y_1 from **y**, the second value x_2 from **x** to the second value y_2 from **y**, and so on. Then, there are n pairs of corresponding values: (x_1, y_1), (x_2, y_2), and so on. Each of these x-y pairs represents a single observation.

The **Pearson (product-moment) correlation coefficient** is a measure of the linear relationship between two features. It's the ratio of the **covariance** of **x** and **y** to the product of their **standard deviations.** It's often denoted with the letter **r** and called **Pearson's r**. You can express this value mathematically with this equation:

$$r = \Sigma_i((x_i - \text{mean}(x))(y_i - \text{mean}(y))) \, (\sqrt{\Sigma_i}(x_i - \text{mean}(x))^2 \, \sqrt{\Sigma_i}(y_i - \text{mean}(y))^2)^{-1}$$

Here, i takes on the values 1, 2, …, n. The mean values of **x** and **y** are denoted with mean(x) and mean(y). This formula shows that if larger x values tend to correspond to larger y values and vice versa, then r is positive. On the other hand, if larger x values are mostly associated with smaller y values and vice versa, then r is negative.

Here are some important facts about the Pearson correlation coefficient:

- The Pearson correlation coefficient can take on any real value in the range $-1 \leq r \leq 1$.
- The maximum value $r = 1$ corresponds to the case when there's a perfect positive linear relationship between **x** and **y**. In other words, larger x values correspond to larger y values and vice versa.
- The value $r > 0$ indicates positive correlation between **x** and **y**.
- The value $r = 0$ corresponds to the case when **x** and **y** are independent.
- The value $r < 0$ indicates negative correlation between **x** and **y**.

- The minimal value r = −1 corresponds to the case when there's a perfect negative linear relationship between **x** and **y**. In other words, larger x values correspond to smaller y values and vice versa.

The above facts can be summed up in the following table:

Pearson's r Value	Correlation Between x and y
equal to 1	perfect positive linear relationship
greater than 0	positive correlation
equal to 0	Independent
less than 0	negative correlation
equal to -1	perfect negative linear relationship

n short, a larger absolute value of r indicates stronger correlation, closer to a linear function. A smaller absolute value of r indicates weaker correlation.

Covariance

Covariance measures the directional relationship between the returns on two assets. A positive covariance means that asset returns move together while a negative covariance means they move inversely. Covariance is calculated by analyzing at-return surprises (standard deviations from the expected return) or by multiplying the **correlation** between the two variables by the **standard deviation** of each variable.

Unlike the correlation coefficient, covariance is measured in units. The units are computed by multiplying the units of the two variables. The variance can take any positive or negative values. The values are interpreted as follows:

- **Positive covariance**: Indicates that two variables tend to move in the same direction.
- **Negative covariance**: Reveals that two variables tend to move in inverse directions.

Formula for Covariance

The covariance formula is similar to the formula for correlation and deals with the calculation of data points from the average value in a dataset. For example, the covariance between two random variables X and Y can be calculated using the following formula (for population):

$$\text{Cov}(X, Y) = \frac{\sum(X_i - \bar{X})(Y_j - \bar{Y})}{n}$$

For a sample covariance, the formula is slightly adjusted:

$$\text{Cov}(X, Y) = \frac{\sum(X_i - \bar{X})(Y_j - \bar{Y})}{n - 1}$$

Where:

- $\mathbf{X_i}$ – the values of the X-variable
- $\mathbf{Y_j}$ – the values of the Y-variable
- $\bar{\mathbf{X}}$ – the mean (average) of the X-variable
- $\bar{\mathbf{Y}}$ – the mean (average) of the Y-variable
- **n** – the number of data points

Covariance vs. Correlation

Covariance and correlation both primarily assess the relationship between variables. The closest analogy to the relationship between them is the relationship between the variance and standard deviation.

Covariance measures the total variation of two random variables from their expected values. Using covariance, we can only gauge the direction of the relationship (whether the variables tend to move in tandem or show an inverse relationship). However, it does not indicate the strength of the relationship, nor the dependency between the variables.

On the other hand, correlation measures the strength of the relationship between variables. Correlation is the scaled measure of covariance. It is dimensionless. In other words, the correlation coefficient is always a pure value and not measured in any units.

The relationship between the two concepts can be expressed using the following formula:

$$\rho(X, Y) = \frac{\mathrm{Cov}(X, Y)}{\sigma_X \sigma_Y}$$

Where:

- **ρ(X,Y)** – the correlation between the variables X and Y
- **Cov(X,Y)** – the covariance between the variables X and Y
- **σ_X** – the standard deviation of the X-variable
- **σ_Y** – the standard deviation of the Y-variable

mean

Mean is a measure of central tendency of a probability distribution along median and mode. It is also referred to as an expected value.

Arithmetic mean is the total of the sum of all values in a collection of numbers divided by the number of numbers in a collection. It is calculated in the following way:

$$\text{Arithmetic mean} = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

Median

Median is a statistical measure that determines the middle value of a dataset listed in ascending order (i.e., from smallest to largest value). The measure divides the lower half from the higher half of the dataset. Along with mean and mode, median is a measure of central tendency.

Although the mean is the most commonly used measure of central tendency for quantitative data, the median can be used instead if the data contains large outliers (data points that are outside the normal data points). The outliers generally skew the mean, while the median is not affected by extreme values. Sometimes the two measures are used simultaneously to determine the value that describes the central value the best.

Calculating Median

The median can be easily found. In some cases, it does not require any calculations at all. The general steps of finding the median include:

1. Arrange the data in ascending order (from the lowest to the largest value).
2. Determine whether there is an even or an odd number of values in the dataset.
3. Considering the results of the previous step, further analysis may follow two distinct scenarios:
4. If the dataset contains an **odd** number of values, the median is a central value that will split the dataset into halves.
5. If the dataset contains an **even** number of values, find the two central values that split the dataset into halves. Then, calculate the mean of the two central values. That mean is the median of the dataset.

To do

Write a python program to calculate median of 10 random points between 1 and 10.

Mode

Mode is the most frequently occurring value in a dataset. Along with mean and median, mode is a statistical measure of central tendency in a dataset. Unlike the other measures of central tendency that are unique to a particular dataset, there may be several modes in a dataset.

How to Find the Mode

No calculations are necessary to find the mode. Simply follow the steps below:

1. Collect and organize the data from a dataset.
2. Determine all the distinct values in a dataset.
3. Count the frequency of occurrence for each distinct value.
4. The most frequent value(s) is the mode.

In addition, it can be easily found using the distribution graph or histogram. Graphically, it is represented as the peak point on the distribution graph or the tallest bar on the histogram.

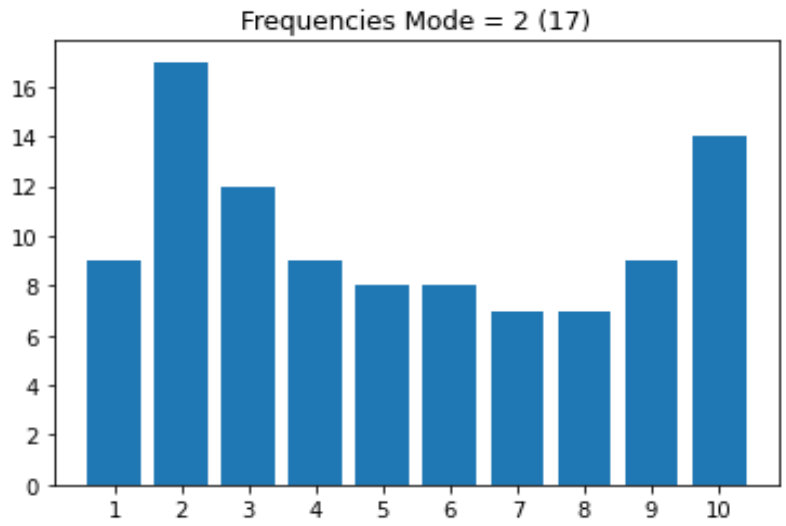
Python code to calculate mode and plot a bar chart from a list of random numbers

```
# program calculate mode from a list of random numbers and plot a bar chart
import random
import matplotlib.pyplot as plt

# count frequencies
freqs = [0]*10

# make x values
x = list(range(1,11))

# make list of numbers
numbers = []
```

```
for i in range(100):
   numbers.append(random.randint(0,9))

# count numbers
for n in set(numbers):
   freqs[n] = numbers.count(n)

# print frequencies
print("frequencies:")
print(freqs)

# calculate mode
mode_freq = max(freqs)
mode = freqs.index(max(freqs))+1

# plot bar chart
plt.bar(x,freqs) # plot bar chart
plt.title("Frequencies Mode = %d (%d)" % (mode,mode_freq))
plt.xticks(x) # set x scale ticks
plt.show() # display plot
```

```
frequencies:

[13, 9, 13, 9, 10, 9, 10, 7, 10, 10]
```

Here is the bar chart:



Standard Deviation

From a statistics standpoint, the standard deviation of a dataset is a measure of the magnitude of deviations between the values of the observations contained in the dataset

Calculating Standard Deviation

We can find the standard deviation of a set of data by using the following formula:

$$SD = \sqrt{\frac{\sum \left(r_i - r_{avg}\right)^2}{n-1}}$$

Where:

- **Ri** – the return observed in one period (one observation in the data set)
- **Ravg** – the arithmetic mean of the returns observed
- **n** – the number of observations in the dataset

By using the formula above, we are also calculating Variance, which is the square of the standard deviation. The equation for calculating variance is the same as the one provided above, except that we don't take the square root.

Difference between regression and correlation

Correlation is a single statistic, or data point, whereas **regression** is the entire equation with all of the data points that are represented **with a** line. **Correlation** shows the **relationship between** the two variables, while **regression** allows us to see how one affects the other.

The difference between these two statistical measurements is that correlation measures the degree of a relationship between two variables (*x* and *y*), whereas regression is how one variable affects another.

Regression is how one variable affects another.

Basically, you need to know when to use correlation vs regression. Use **correlation** for a quick and simple summary of the direction and strength of the relationship between two or more numeric variables. Use **regression** when you're looking to predict, optimize, or explain a number response between the variables (how *x* influences *y*).

Calculating and Plotting Correlation

We first make data points x and y

```
# make point x
x = np.arange(10, 20)
# make point y
y = np.array([2, 1, 4, 5, 8, 12, 18, 25, 96, 48])
# print out points
print(x)
print(y)
```

```
x =  [10 11 12 13 14 15 16 17 18 19]
y =  [ 2  1  4  5  8 12 18 25 96 48]
```

Next we calculate correlation by formula:

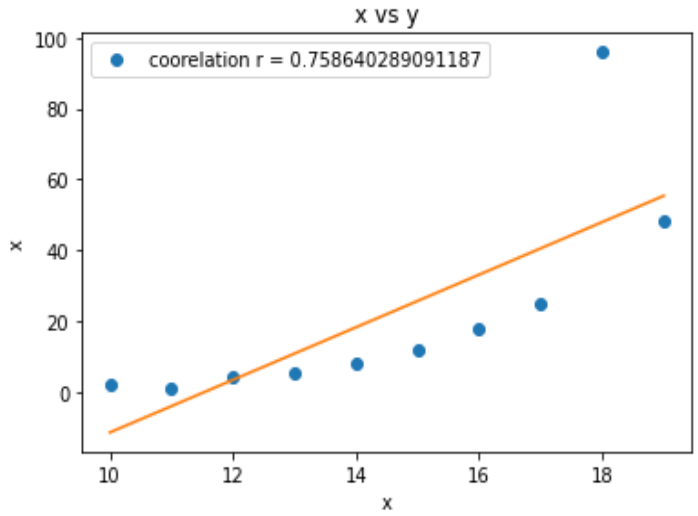
$r = \Sigma_i((x_i - mean(x))(y_i - mean(y))) \, (\sqrt{\Sigma_i(x_i - mean(x))^2} \, \sqrt{\Sigma_i(y_i - mean(y))^2})^{-1}$

```
# calculate correlation by formula
# r = Σi((xi – mean(x))(yi – mean(y))) (√Σi(xi – mean(x))² √Σi(yi – mean(y))²)⁻¹
top = sum((x - x.mean())*(y - y.mean()))
bottom =  (math.sqrt(sum((x - x.mean())*(x - x.mean()))) * math.sqrt(sum((y - y.mean())*(y - y.mean()))))
r = top / bottom
print(r)
```

Finally we plot the points and regression line and display the calculated correlation value r.

```
# plot points and regression line
plt.plot(x, y, 'o', label = "correlation r = " + str(r))
m, b = np.polyfit(x, y, 1)
plt.plot(x, m*x + b)
plt.title('x vs y')
plt.ylabel('x')
plt.xlabel('x')
plt.legend()
plt.show()
```

Our scatter points and regression line plot is as follows



NumPy has the function **corrcoef(),** that returns a matrix of Pearson correlation coefficients for input arrays x and y

```
# make correlation matrix
r = np.corrcoef(x, y)

# print correlation matrix
print(r)
```

```
coorelation matrix =
[[1.         0.75864029]
 [0.75864029 1.        ]]
```

The values on the main diagonal of the correlation matrix (upper left and lower right) are equal to 1. The upper left value corresponds to the correlation coefficient for x and x, while the lower right value is the correlation coefficient for y and y. They are always equal to 1.

The lower left and upper right values of the correlation matrix are equal and both represent the **Pearson correlation coefficient** for x and y. In this case, it's approximately 0.76.

We can plot a pair plot to view the x and y points as a scatter plot the correlation trends as regression lines. Regression line are different from correlation. A regression line is a continues line of points representing the slope of the points where correlation is a single value representing the estimated slope of the line.

Calculating correlation with pandas data frame

```
# make a data frame of x-y values
df = pd.DataFrame({'x': x, 'y': y})
print(df)
```

```
    x   y
0  10   2
1  11   1
2  12   4
3  13   5
4  14   8
5  15  12
6  16  18
7  17  25
8  18  96
9  19  48
```

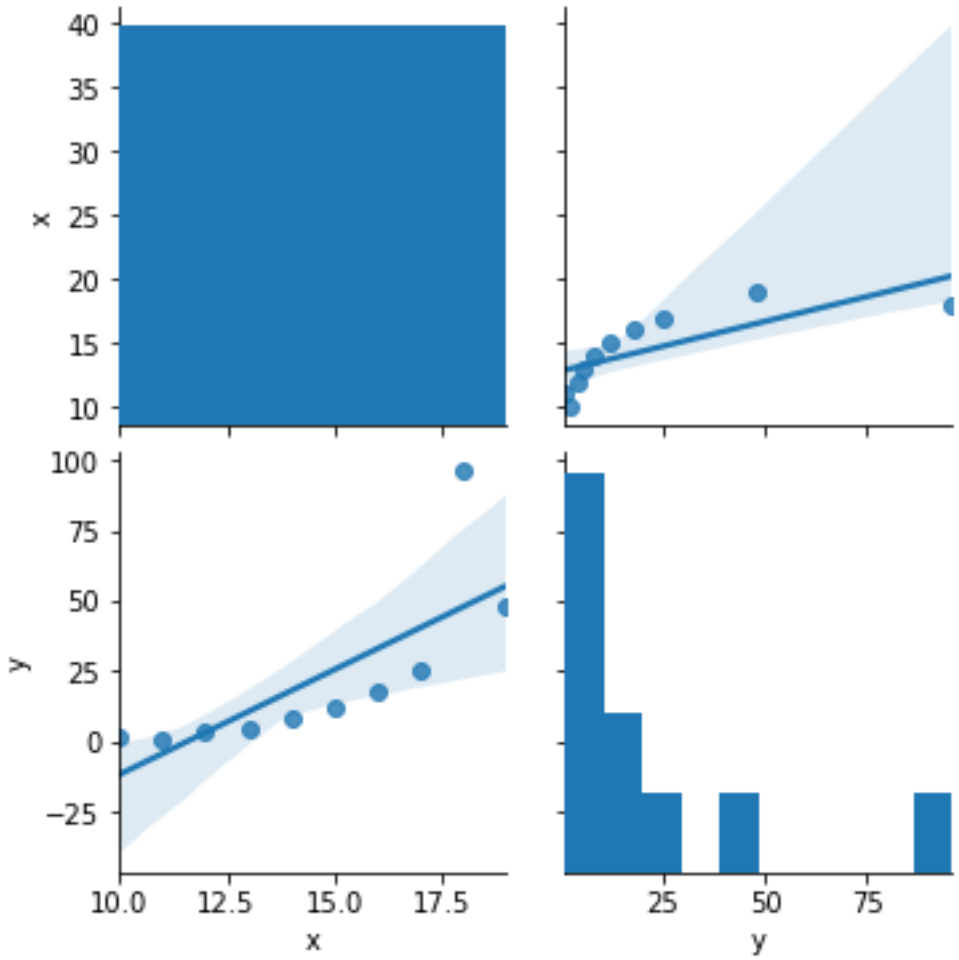
Calculate correlation using the panda s corr function

```
# df coorelation
coor = df.corr()
print(coor)
```

```
          x         y
x  1.00000  0.75864
y  0.75864  1.00000
```

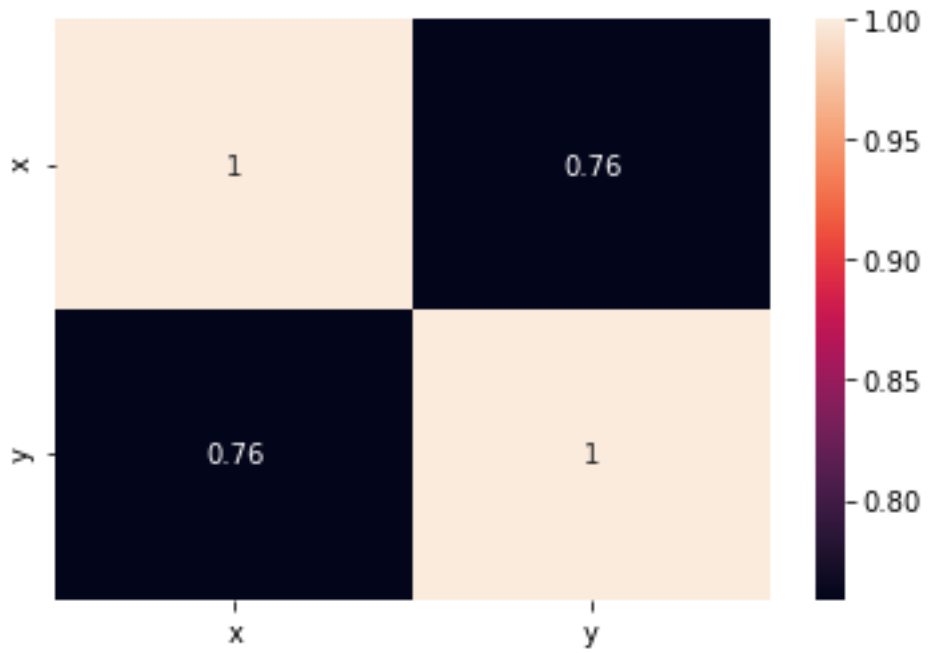
We can visualize correlation with a combined scatter regression pair plot.

```
# plot pair plot with regression line
sns.pairplot(df,kind="reg")
plt.show()
```



We can visualize the correlation values places in a heat map

```
# calculate the correlation matrix
corr = df.corr()

# set heatmap back ground colors
corr.style.background_gradient(cmap='coolwarm')

# plot the heatmap
sns.heatmap(corr, xticklabels=corr.columns,yticklabels=corr.columns,annot = True)
plt.show()
```



To do:

make a data frame with 3 or 4 columns and 10 rows of data and print out the correlation matrix. plot the pair plot and correlation heat map

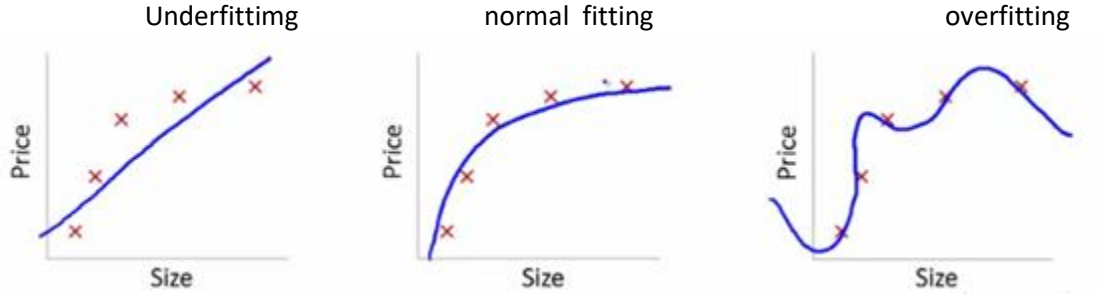
CORRELATION HOMEWORK

Make a data frame with x and columns and 10 rows of data and calculate and print out the correlation, covariance, mean, mode, standard deviation and variance. Plot a scatter plot of the points and a regression line. In the plot display and label the correlation, covariance, mean, mode, standard deviation and variance. Name your python file coorelationhomework.py

CROSS VALIDATION

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The limited sample is used in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

Cross-validation is mainly used as a way to check for under or over-fit.
In the following plots we have underfitting, fitting and overfitting.



The first plot represents a linear relationship and has a high error from training data point and is an example of underfitting. The model fails to capture the underlining trend of data.

The second plots represents normal fitting has a low training error and a generalization of the relationship between price and size. The curve follows the points very tightly.

In the third plot we have almost zero training error, but is too sensitive and captures random patterns that are only in the present training set but not present in other datasets. This is an example overfitting and there could be a high deviation between training sets and actual data.

A common practice in data science competitions is to iterate over various models to find a better performing model. However, it becomes difficult to distinguish whether this improvement in score is coming because we are capturing the relationship better, or we are just over-fitting the data. To find the right answer for this question, we use **validation** techniques. This method helps us in achieving more generalized relationships.

To **avoid over-fitting**, we have to define two different sets : a **training set** X_train, y_train which is used for learning the parameters of a predictive model, and a **testing set** X_test, y_test which is used for evaluating the fitted predictive model.

We can now quickly sample a training set while holding out a percentage of the data for testing (evaluating) our classifier:

However, by defining these two sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, test) sets.

A solution is to **split the whole data several consecutive times in different train set and test set**, and to return the averaged value of the prediction scores obtained with the different sets. Such a procedure is called **cross-validation**. This approach can be **computationally expensive, but does not waste too much data** (as it is the case when fixing an arbitrary test set), which is a major advantage in problem such as inverse inference where the number of samples is very small.

k-fold cross-validation

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
    1. Take the group as a hold out or test data set
    2. Take the remaining groups as a training data set
    3. Fit a model on the training set and evaluate it on the test set
    4. Retain the evaluation score and discard the model
4. Summarize the skill of the model using the sample of model evaluation scores

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times.

This approach involves randomly dividing the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining k − 1 folds.

The results of a k-fold cross-validation run are often summarized with the mean of the model skill scores. It is also good practice to include a measure of the variance of the skill scores, such as the standard deviation or standard error.

Configuration of k

The k value must be chosen carefully for your data sample.

A poorly chosen value for k may result in a mis-representative idea of the skill of the model, such as a score with a high variance (that may change a lot based on the data used to fit the model), or a high bias, (such as an overestimate of the skill of the model).

Three common tactics for choosing a value for k are as follows:

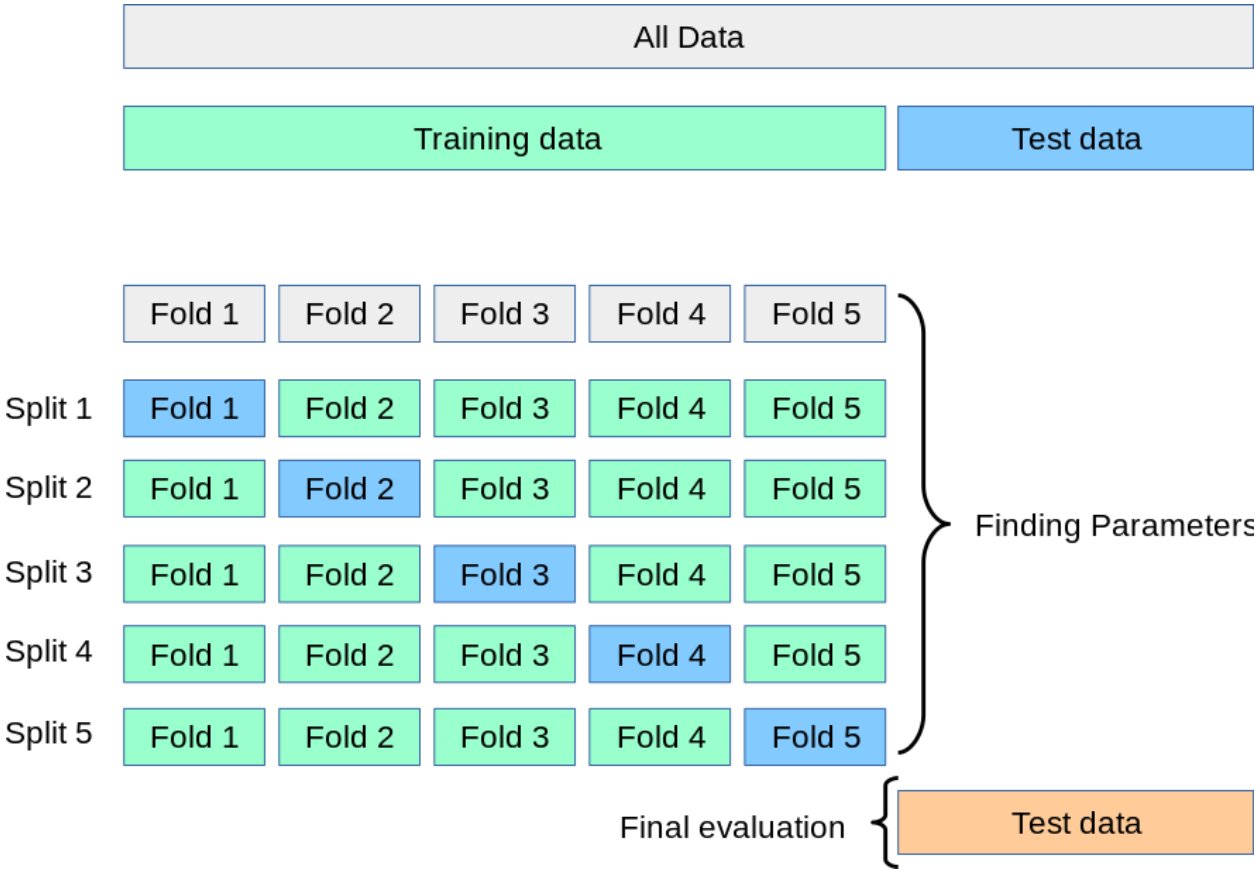
- **Representative**: The value for k is chosen such that each train/test group of data samples is large enough to be statistically representative of the broader dataset.
- **k=10**: The value for k is fixed to 10, a value that has been found through experimentation to generally result in a model skill estimate with low bias a modest variance.
- **k=n**: The value for k is fixed to n, where n is the size of the dataset to give each test sample an opportunity to be used in the hold out dataset. This approach is called leave-one-out cross-validation.

The choice of k is usually 5 or 10, but there is no formal rule. As k gets larger, the difference in size between the training set and the resampling subsets gets smaller. As this difference decreases, the bias of the technique becomes smaller

A value of k=10 is very common in the field of applied machine learning, and is recommend if you are struggling to choose a value for your dataset.

To summarize, there is a bias-variance trade-off associated with the choice of k in k-fold cross-validation. Typically, given these considerations, one performs k-fold cross-validation using k = 5 or k = 10, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance.

If a value for k is chosen that does not evenly split the data sample, then one group will contain a remainder of the examples. It is preferable to split the data sample into k groups with the same number of samples, such that the sample of model skill scores are all equivalent



Evaluating a Machine Learning model can be quite tricky. Usually, we split the data set into training and testing sets and use the training set to train the model and testing set to test the model. We then evaluate the model performance based on an error metric to determine the accuracy of the model. This method however, is not very reliable as the accuracy obtained for one test set can be very different to the accuracy obtained for a different test set. *K-fold Cross Validation(CV)* provides a solution to this problem by dividing the data into folds and ensuring that each fold is used as a testing set at some point.

K-Fold CV is where a given data set is split into a ***K*** number of sections/folds where each fold is used as a testing set at some point. Lets take the scenario of 5-Fold cross validation(K=5). Here, the data set is split into 5 folds. In the first iteration, the first fold is used to test the model and the rest are used to train the model. In the second iteration, 2nd fold is used as the testing set while the rest serve as the training set. This process is repeated until each fold of the 5 folds have been used as the testing set.

Sklearn kFold

Sklearn has the K-Folds cross-validator **model** located in **sklearn.model_selection.** It provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default). Each fold is then used once as a validation while the k - 1 remaining folds form the training set. Sklearn also has the **cross_val_score** and **cross_val_predict** modules located in sklearn.model_selection. **cross_val_score** return the scores of the kFold training sets where as **cross_val_predict** return the prediction of the kFold training sets. Both receive a classifier model and X and Y data points.

KFold test program

We first make the necessary imports as usual

```
from sklearn.model_selection import KFold
import numpy as np
import sklearn.linear_model.logistic
from sklearn.datasets import make_classification
```

We will use a logistic regression model because it is very good in classifying binary data . We use the sklearn **make_classification function** to make our X and y dataset**.** We make 100 samples with 5 features and 1 binary target .

We then use the KFold function to calculate the different K Fold training sets.

```
# make 100 training samples with 5 features
X, y = make_classification(n_samples=100, n_features=5)

# store scores
scores = []
predictions = []

# make LinearRegression model
model = LogisticRegression()

# make KFold with 10 splits
cv = KFold(n_splits=10, random_state=42, shuffle=True)
```

For each folded training set, we print out the indexes, get the training data from the indexes then use the classifier to calculate the sore. and predictions. We store the score and predictions in a list

```
# for each training set produced
for train_index, test_index in cv.split(X):

   # print train indexes
   print("Train Index:\n", train_index, "\n")

   # print test indexes
   print("Test Index:\n", test_index)

   # get the data from the indexes
   X_train, X_test, y_train, y_test = X[train_index], X[test_index], y[train_index], y[test_index]

   # use the model to classify the data
   model.fit(X_train, y_train)

   # calculate score accuracy using the model
   score = model.score(X_test, y_test)
```

```
    # print score
    print("score: ",score)

    # store score
    scores.append(score)

     # calculate predictions using model
     prediction = model.predict(X_test)

      # print predictions
      print("predistion: ",prediction)

       # store predictions
       predictions.append(prediction)



# print scores
print("KFold scores:\n",scores)

# print mean and std of scores
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))

# print predictions
print("kFold predictions:\n",predictions)
```

```
KFold scores:

 [0.7, 1.0, 1.0, 0.9, 0.9, 1.0, 0.9, 1.0, 0.7, 0.9]

Accuracy: 0.900 (0.110)
```

```
# print mean and std of predictions
print('Accuracy: %.3f (%.3f)' % (np.mean(predictions), np.std(predictions)))
```

```
KFold predictions:

 [array([0, 0, 0, 1, 0, 1, 0, 0, 1, 1]),
  array([0, 0, 0, 1, 1, 1, 1, 0, 1, 1]),
  array([0, 1, 0, 1, 0, 1, 1, 1, 0, 1]),
  array([1, 0, 0, 0, 1, 1, 0, 1, 1, 1]),
  array([0, 0, 1, 0, 0, 1, 1, 1, 1, 0]),
  array([1, 1, 1, 1, 0, 1, 1, 0, 1, 0]),
  array([1, 0, 1, 0, 0, 1, 0, 1, 0, 1]),
  array([0, 1, 0, 1, 0, 1, 1, 0, 1, 0]),
  array([1, 1, 0, 0, 0, 0, 0, 0, 0, 0]),
  array([0, 0, 0, 0, 0, 1, 0, 0, 0, 1])]

Accuracy: 0.480 (0.500)
```

We use the sklearn module to do the same thing as above. to calculate and print the scores.

```
# using cross val score
print("using cross val score")

# use cross validation to calculate score using model and dataset
scores = cross_val_score(model, X, y, cv=10)

# print out scores
print("Cross Val scores:\n",scores)

# report performance
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
using cross val score
Cross Val scores:
 [1.  1.  0.8 1.  0.7 0.9 0.9 0.8 0.9 1. ]
Accuracy: 0.900 (0.100)
```

We use the sklearn module to calculate and print the predictions

```
# using cross val predict
print("using cross val score")

# using cross val predict
predictions = cross_val_predict(model, X, y, cv=10)

# print predictions
print("Cross Val predictions:\n",predictions)
# report performance
print('Accuracy: %.3f (%.3f)' % (np.mean(predictions), np.std(predictions)))
```

```
using cross val predictions
Cross Val predictions:
 [0 1 1 1 0 1 1 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0
  0 0 0 1 1 0 1 1 1 1 0 1 1 1 1 0 1 0 1 0 1 0 1 1 0 0 0
  0 1 0 1 0 1 0 1 1 1 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1 0 1
  1 0 1 0 1 0 0 0 0 1 0 1 1 1 0 1 0 0 1]
Accuracy: 0.500 (0.500)
```

CROSS VALIDATION HOMEWORK

UAE THE same data set from Homework 1 Do KFold on it. on it. from the kfold select the best performing data set then use cross val score and cross val predict in the selected x and y. compare if you get the same results. Name your python file crossvalidationhomework.py