Coverage-based testing

UW CSE P 504
Today

- Recap: Git bisect exercise
- Software testing 101
- Test adequacy: structural code coverage
  - Statement coverage
  - Decision coverage
  - Condition coverage
  - Modified condition and decision coverage (MCDC)
- Discussion of papers
- In-class activity: code coverage
Recap: git bisect

- Git bisect run-time complexity is ...?
Recap: git bisect

- Git bisect run-time complexity is always $O(\log(n))$
Recap: git bisect

- Git bisect run-time complexity is always $O(\log(n))$

- Git revert = add inverted commit; git reset = remove a commit
Recap: git bisect

- Git bisect run-time complexity is always $O(\log(n))$

- Git revert = add inverted commit; git reset = remove a commit

- `git rev-list v1.0.0..HEAD` (or HEAD ^v1.0.0)
Recap: git bisect

Questions

● How could the developers improve the build or testing infrastructure to notice test failures in the future?

● Which git command can you use to undo a defect-inducing commit? Briefly explain what problem may generally occur when undoing a commit and what best practices mitigate this problem.

● Can you undo the defect-inducing commit using the proposed git command?
Recap: git bisect

Questions

- How could the developers improve the build or testing infrastructure to notice test failures in the future?
- Which git command can you use to undo a defect-inducing commit? Briefly explain what problem may generally occur when undoing a commit and what best practices mitigate this problem.
- Can you undo the defect-inducing commit using the proposed git command?

Meta-level discussion

- Is Git bisect a realistic choice for the JavaParser example?
- I don’t use Java, so why should I care?
- Forum participation is great!
Software testing 101
Testing vs. debugging

double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;

    int i = 0;
    while (i<n) {
        sum = sum + nums[i];
        i = i + 1;
    }

    double avg = sum * n;
    return avg;
}
Testing vs. debugging

Testing: is there a failure?

@Test
class Main {
    public void testAvg() {
        double[] nums = {1.0, 2.0, 3.0};
        double actual = Math.avg(nums);
        double expected = 2.0;
        assertEquals(expected, actual, EPS);
    }
}
Testing vs. debugging

Testing: is there a failure?

```
@Test
public void testAvg() {
    double nums =
        new double[]{1.0, 2.0, 3.0});
    double actual = Math.avg(nums);
    double expected = 2.0;
    assertEquals(expected, actual, EPS);
}

testAvg failed: 2.0 != 18.0
```
Testing vs. debugging

Testing: is there a failure?
@Test
public void testAvg() {
    double nums =
        new double[] {1.0, 2.0, 3.0};
    double actual = Math.avg(nums);
    double expected = 2.0;
    assertEquals(expected, actual, EPS);
}

testAvg failed: 2.0 != 18.0

Debugging: where is the defect?
how to fix the defect?

double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    int i = 0;
    while (i<n) {
        sum = sum + nums[i];
        i = i + 1;
    }
    double avg = sum * n;
    return avg;
}
Testing vs. debugging

Testing: is there a failure?

```java
@Test
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;
  int i = 0;
  while (i < n) {
    sum = sum + nums[i];
    i = i + 1;
  }
  double avg = sum * n / n;
  return avg;
}
```

testAvg failed: 2.0 != 18.0

Debugging: where is the defect? how to fix the defect?
Software testing is unsound

“Program testing can … show the presence of bugs but never … show their absence.”
(Edsger W. Dijkstra)
Software testing is unsound

“Program testing can … show the presence of bugs but never … show their absence.”
(Edsger W. Dijkstra)

- A good test is one that fails because of a defect.

How do we come up with good tests?
Devising good tests

- Partition input into different behaviors
- One test for each behavior
- Use heuristics for partitioning (= for choosing tests)
  - Take inspiration from the theory of revealing subdomains
Two strategies: black box vs. clear box

Black box testing
- The system is a black box (can’t see inside).
- No knowledge about the internals of a system.
- Create tests solely based on the specification (e.g., input/output behavior).

Clear box testing
- Knowledge about the internals of a system.
- Create tests based on these internals (e.g., exercise a particular part or path of the system).
Clear box testing example

Some subdomains are not evident from the specification

```java
boolean[] primeTable = new boolean[CACHE_SIZE];

boolean isPrime(int x) {
    if (x < CACHE_SIZE) {
        return primeTable[x];
    } else {
        for (int i = 2; i < x/2; i++) {
            if (x%i == 0)
                return false;
        }
        return true;
    }
}
```

Subdomain boundary (execution difference) at: \( x = \text{CACHE\_SIZE} \)
Clear box testing tradeoffs

Advantages

● Provides an important source of boundaries
● Has an objective test quality metric: coverage

Disadvantages

● Tests may have same bugs as implementation
  ○ Buggy code tricks you into complacency as soon as you read it
Unit testing, integration testing, system testing

**Unit testing**
- Does each unit work as specified?

**Integration testing**
- Do the units work when put together?

**System testing**
- Does the system work as a whole?
Unit testing, integration testing, system testing

Unit testing
● Does each unit work as specified?

Integration testing
● Do the units work when put together?

System testing
● Does the system work as a whole?

Our focus: unit testing
Unit testing

● A **unit** is the *smallest testable part* of the software system (e.g., a method or a function).

● **Goal:** Verify that each software unit performs as specified.

● **Focus:**
  ○ Individual unit
    ■ Not the interactions between units
    ■ Not dependences of the unit
  ○ Usually input/output relationships
Software testing is unsound

“Program testing can … show the presence of bugs but never … show their absence.”
(Edsger W. Dijkstra)

- A good test is one that fails because of a defect.

When should we stop testing if no (new) test fails?
Test effectiveness

Ratio of detected defects is all that matters!

Problem
- The set of defects is unknowable.

Solution
- Use a proxy metric, for example code coverage.

Adequacy = score according to the proxy metric
Adequate = 100% score
Test adequacy: structural code coverage
Structural code coverage: motivating example

Average of the absolute values of an array of doubles

```java
public double avgAbs(double ... numbers) {

    // We expect the array to be non-null and non-empty
    if (numbers == null || numbers.length == 0) {
        throw new IllegalArgumentException("Array numbers must not be null or empty!");
    }

    double sum = 0;
    for (int i=0; i<numbers.length; ++i) {
        double d = numbers[i];
        if (d < 0) {
            sum -= d;
        } else {
            sum += d;
        }
    }

    return sum/numbers.length;
}
```

What tests should we write for this method?
### Cobertura’s line coverage report

<table>
<thead>
<tr>
<th>Classes in this File</th>
<th>Line Coverage</th>
<th>Branch Coverage</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg</td>
<td>100%</td>
<td>10/10</td>
<td>6</td>
</tr>
</tbody>
</table>

```java
package avg;

public class Avg {

    /**
     * Compute the average of the absolute values of an array of doubles
     */
    public double avgAbs(double ... numbers) {
        // We expect the array to be non-null and non-empty
        if (numbers == null || numbers.length == 0) {
            throw new IllegalArgumentException("Array numbers must not be null or empty!");
        }

        double sum = 0;
        for (int i=0; i<numbers.length; ++i) {
            double d = numbers[i];
            if (d < 0) {
                sum -= d;
            } else {
                sum += d;
            }
        }
        return sum/numbers.length;
    }
}
```
public double avgAbs(double ... numbers) {

    // We expect the array to be non-null and non-empty
    if (numbers == null || numbers.length == 0) {
        throw new IllegalArgumentException("Array numbers must not be null or empty!");
    }

double sum = 0;
for (int i=0; i<numbers.length; ++i) {
    double d = numbers[i];
    if (d < 0) {
        sum -= d;
    } else {
        sum += d;
    }
}
return sum/numbers.length;
}
Control flow graph

Entry point

- $a == \text{null} \ || \ a.\text{length}==0$
  - true: throw new IllegalArgumentException("Array a must not be null or empty!")
  - false: sum = 0

- $i = 0$

- $i < a.\text{length}$
  - false: return sum/a.\text{length}
  - true: $num = a[i]$

- $num < 0$
  - false: sum += num
  - true: sum -= num

- $++i$
public double avgAbs(double ... numbers) {

    // We expect the array to be non-null and non-empty
    if (numbers == null || numbers.length == 0) {
        throw new IllegalArgumentException("Array numbers must not be null or empty!");
    }

    double sum = 0;
    for (int i=0; i<numbers.length; ++i) {
        double d = numbers[i];
        if (d < 0) {
            sum -= d;
        } else {
            sum += d;
        }
    }

    return sum/numbers.length;
}
Statement coverage

- Every **statement** in the program must be executed at least once.
Statement coverage

Entry point

a==null || a.length==0
true
false

sum = 0
i = 0

i<a.length
false
true

num = a[i]

num < 0
false
true

num += num
sum -= num
++i

return sum/a.length

throw new IllegalArgumentException("Array a must not be null or empty!")

Exceptional exit

Normal exit
Statement coverage

- Every **statement** in the program must be executed at least once.
- This is **node coverage** in the control-flow graph (CFG).

```
Entry point

a == null || a.length == 0

false

sum = 0
i = 0
true

i < a.length

false

num = a[i]
false

num < 0
true

sum -= num
true

++i
false

sum += num
false

throw new IllegalArgumentException("Array a must not be null or empty!")

Exceptional exit

Normal exit
```
100% coverage

- Usually unachievable (dead code)
- Prohibitively expensive
Statement coverage is not enough

```c
int min(int a, int b) {
    int result = a;
    if (a <= b) {
        result = a;
    }
    return result;
}
```

Consider any test with $a \leq b$, e.g., $\text{min}(1, 2)$

- It executes every instruction
- It misses the defect
Branch coverage

- Every **branch** in the program must be executed at least once.
- A branch is each outcome of a conditional statement’s test: if, for, while, ...
Branch coverage

Entry point

(a==null || a.length==0) true

sum = 0
i = 0

i<a.length false

i=a.length true

return sum/a.length

(a==null || a.length==0) false

Throw new IllegalArgumentException("Array a must not be null or empty!") Exceptional exit

sum = 0
i = 0

i<a.length false

i=a.length true

return sum/a.length

num < 0 false

num = a[i]

num < 0 true

num = a[i]

sum += num

num < 0 false

num = a[i]

sum -= num

++i

Normal exit
Branch coverage

Entry point

\[ a == \text{null} || a.\text{length} == 0 \]

\[ \text{sum} = 0 \]
\[ i = 0 \]

\[ i < a.\text{length} \]

\[ \text{num} = a[i] \]

\[ \text{num} < 0 \]

\[ \text{sum} -= \text{num} \]

\[ ++i \]

\[ \text{return sum/a.\text{length}} \]

throw new IllegalArgumentException("Array a must not be null or empty!")

Exceptional exit

Normal exit
Branch coverage

- Every **branch** in the program must be executed at least once.
- This is **edge coverage** in the CFG.
Branch coverage is not enough

```c
int quadrant(int x, int y) {
    int answer;
    if (x >= 0)
        answer = 1;
    else
        answer = 2;
    if (y < 0)
        answer = 4;
    return answer;
}
```

Consider a suite with two test inputs: (2,-2) and (-2,2)

- Achieves 100% branch coverage
- Misses the bug
Branch coverage = decision coverage

- Every decision in the program must take on all possible outcomes (true/false) at least once.
Terminology: conditions and decisions

- **Condition**: an atomic boolean expression
  - contains no smaller boolean expressions
- **Decision**: a maximal boolean expression in the source code
  - decision = one or more conditions joined with logical connectors
- **Example**: if \((a \lor b)\) { … }
  - “\(a\)” and “\(b\)” are *conditions*.
  - “\(a \lor b\)” is a *decision*. 
Decision coverage

```java
Decision entry point:

a == null || a.length == 0

sum = 0
i = 0

i < a.length

num = a[i]

num < 0

sum += num
num = a[i]
++i

if (true) throw new IllegalArgumentException("Array a must not be null or empty!")

return sum / a.length

Normal exit

Exceptional exit
```
Decision coverage = branch coverage

Entry point

\[a == \text{null} \text{ || } a.\text{length} == 0\]

sum = 0
i = 0

i < a.length

num = a[i]

num < 0

sum -= num
++i

true
false
false
true
false
true

return sum/a.length

throw new IllegalArgumentException("Array a must not be null or empty!")

Exceptional exit

Normal exit
Condition coverage

- Every **condition** in the program must take on all possible outcomes (true/false) at least once.
Condition coverage

Entry point

\[ a == \text{null} \text{ or } a.length == 0 \]

- true: throw new IllegalArgumentException("Array a must not be null or empty!")
- false: sum = 0
- false: i = 0

\[ i < a.length \]

- true: num = a[i]
- false: return sum/a.length

\[ num < 0 \]

- true: sum -= num
- false: sum += num

++i
Condition coverage

- Every **condition** in the program must take on all possible outcomes (true/false) at least once.
- Think of another explanation for condition coverage, in terms of the **binary**.
Structural code coverage: subsumption

Given two coverage criteria A and B, A subsumes B iff satisfying A implies satisfying B

Subsumption relationships:

1. Does statement coverage subsume branch (decision) coverage?
2. Does branch (decision) coverage subsume statement coverage?

If so, give a brief argument.

If not, give a counterexample.

In: if (\(a \text{ || } b\)) { … }, “\(a \text{ || } b\)” is a decision.
Structural code coverage: subsumption

Given two coverage criteria A and B, 
A subsumes B iff satisfying A implies satisfying B

Subsumption relationships:

1. **Statement** coverage does not subsume **decision** coverage
2. **Decision** coverage subsumes **statement** coverage

In: if (a || b) { … },
“a || b” is a decision.
Structural code coverage: subsumption

Given two coverage criteria A and B, **A subsumes B iff satisfying A implies satisfying B**

Subsumption relationships:

1. Does decision coverage subsume condition coverage?
2. Does condition coverage subsume decision coverage?

If so, give a brief argument.

If not, give a counterexample.

In: if \((a \| b)\) { … }

- “\(a \| b\)” is a decision.
- “\(a\)” and “\(b\)” are conditions.
Structural code coverage: subsumption

Given two coverage criteria A and B, 
**A subsumes B iff satisfying A implies satisfying B**

Subsumption relationships:

1. **Decision** coverage does not subsume **condition** coverage
2. **Condition** coverage does not subsume **decision** coverage

In: if \((a || b)\) { … }

- “a \(\|\ b\)” is a decision.
- “a” and “b” are conditions.
Decision coverage vs. condition coverage

4 possible tests for the decision $a \mid b$:

1. $a = 0, b = 0$
2. $a = 0, b = 1$
3. $a = 1, b = 0$
4. $a = 1, b = 1$

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
<th>$a \mid b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Satisfies condition coverage but not decision coverage

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
<th>$a \mid b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Does not satisfy condition coverage but decision coverage

Neither coverage criterion subsumes the other!
MCDC: Modified condition and decision coverage

- **Decision coverage** (every decision is both true & false)
- **Condition coverage** (every condition is both true & false)
- Each condition independently affects its decision’s outcome.
  Hold other conditions fixed, vary that condition, decision changes.

Required for safety critical systems (DO-178B/C)
MCDC: an example

if (a | b)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Which tests (combinations of a and b) satisfy MCDC?

MCDC
- Decision coverage
- Condition coverage
- Each condition independently affects its decision
MCDC: an example

if (a | b)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

MCDC
- Decision coverage
- Condition coverage
- Each condition independently affects its decision

MCDC is still cheaper than testing all possible combinations.
MCDC: another example

```
if (a || b)
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

MCDC

- Decision coverage
- Condition coverage
- Each condition independently affects its decision

Why is this example different?
MCDC: another example

```java
if (a || b)
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>Outcome</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---------</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>--</td>
<td>1</td>
</tr>
</tbody>
</table>

**MCDC**
- **Decision** coverage
- **Condition** coverage
- Each **condition independently affects** its decision

Short-circuiting operators may not evaluate all conditions.
MCDC: yet another example

```c
if (!a) { ... if (a || b) ... }
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

MCDC:
- Decision coverage
- Condition coverage
- Each condition independently affects its decision

What about this example?
MCDC: yet another example

```java
if (!a) { ... if (a || b) ... }
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**MCDC**
- Decision coverage
- Condition coverage
- Each condition independently affects its decision

Not all combinations of conditions may be possible.
MCDC: complex expressions

Provide an MCDC-adequate test suite for:

1. $a \mid b \mid c$
2. $a \& b \& c$

MCDC
- Decision coverage
- Condition coverage
- Each condition independently affects its decision
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
### a & b & c

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
A program with a loop has an infinite number of paths.
Other notions of coverage ("program spectra")

Example use: run a program with dates before & after Y2K
Structural code coverage: summary

- Code coverage is easy to compute.
- Code coverage has an intuitive interpretation.
- Code coverage is used in industry: Code coverage at Google
- Code coverage itself is not sufficient!
Mutation-based testing: teaser

A better test suite detects more real defects

- One proxy metric: code coverage
- Another proxy metric: detection of fake defects (“mutants”)

Mutant = a small program change

\[
a = b + c; \quad \implies \quad a = b - c; \\
a = b + c; \quad \implies \quad a = b \times c; \\
a = b + c; \quad \implies \quad a = b + 0; \\
a = b + c; \quad \implies \quad a = 1 + c;
\]

Intuitive argument about mutants and real defects:

a mutant might be defective
∧ detecting more mutants means detecting more fake defects
\[\therefore\] detecting more mutants means the test suite is better
Measuring mutation score

Input: program and test suite

1. Apply *mutation operators* to the program to create many mutants
   Each mutant is a variation of the program
2. For each mutant, run the test suite
3. Count the number of test runs that fail
   If the test run failed, then the test suite “detected” or “killed” the mutant.
4. Mutation score = \( \frac{\text{detected mutants}}{\text{all mutants}} \)
Discussion of papers
Ice cream causes murder

![Graph showing the correlation between ice cream sales and violent crime index.](image)
Murder causes ice cream sales
1. **Stress and Emotional Response**: During times of increased violent crime, individuals within a community may experience heightened levels of stress, fear, or anxiety due to concerns for their safety or the safety of their loved ones. This elevated emotional state might lead some people to seek out comfort foods as a way to cope with their emotions.

2. **Psychological Response**: Ice cream, being a popular comfort food, could serve as a psychological coping mechanism for individuals experiencing stress. The act of consuming ice cream may trigger the release of neurotransmitters such as serotonin and endorphins, which are associated with feelings of pleasure and relaxation. As a result, some individuals may be more inclined to purchase and consume ice cream during periods of heightened stress or emotional distress.

3. **Behavioral Patterns**: An increase in violent crime may lead to changes in individuals' behavior, including their consumption habits. People may be more likely to indulge in comfort foods like ice cream as a way to temporarily alleviate their stress or anxiety. This could result in a noticeable uptick in ice cream sales within communities experiencing higher levels of violent crime.

4. **Cultural Factors**: Cultural norms and societal responses to stress or trauma may play a role in shaping individuals' behaviors. In some communities, the consumption of certain foods, such as ice cream, may be ingrained as a socially acceptable way to cope with difficult emotions or situations.
Ice cream and murder
Correlation or causation?

Number of people who drowned by falling into a pool correlates with Films Nicolas Cage appeared in

Correlation: 66.6% (r=0.666004)

Data sources: Centers for Disease Control & Prevention and Internet Movie Database
Correlation or causation?

**Kerosene used in Cuba** correlates with **Google searches for 'cia hotline'**

- **Volume of kerosene used consumed in Cuba in millions of barrels per day**
  - Source: Energy Information Administration

- **Relative volume of Google searches for 'cia hotline' (Worldwide, without quotes)**
  - Source: Google Trends

2004-2021, r=0.992, r²=0.984, p<0.01 • tylervigen.com/spurious/correlation/1102
Correlation or causation?

How provocative Matt Parker's YouTube video titles are correlates with Value of the Victoria's Secret Annual Fantasy Bra.

- How provocative Stand-up Maths YouTube video titles are, as rated by an AI. Source: AI analysis of Stand-up Maths YouTube video titles.
- Value of the Victoria's Secret Annual Fantasy Bra. Source: Wikipedia. 2011-2019, r=0.858, r^2=0.737, p<0.01. tylervigen.com/spurious/correlation/4629
Correlation or causation?

Pirate attacks globally correlates with Google searches for 'download firefox'

[Line graph showing the correlation between pirate attacks and Google searches for 'download firefox']

- Global Pirate Attack Count · Source: Statista
- Relative volume of Google searches for 'download firefox' (Worldwide, without quotes) · Source: Google Trends

2009-2022, r=0.974, r²=0.949, p<0.01 · tylervigen.com/spurious/correlation/2204
Correlation or causation?

Popularity of the 'its wednesday my dudes' meme correlates with Boeing's stock price (BA)

- Relative volume of Google searches for 'its wednesday my dudes' (without quotes, in the United States) · Source: Google Trends
- Opening price of The Boeing Company (BA) on the first trading day of the year · Source: LSEG Analytics (Refinitiv)

2006-2023, r=0.877, r²=0.766, p<0.01 · tylervigen.com/spurious/correlation/4977
Ice cream causes polio

Benjamin Sandler showed a correlation between sugar consumption and polio

- By country
- By month
  - Polio is at its height in summer when sugar intake is highest (ice cream, soft drinks, ...)

Articles in the 1940s; 1951 book

Reduced ice cream sales by over 1 million gallons per week
Discussion of papers