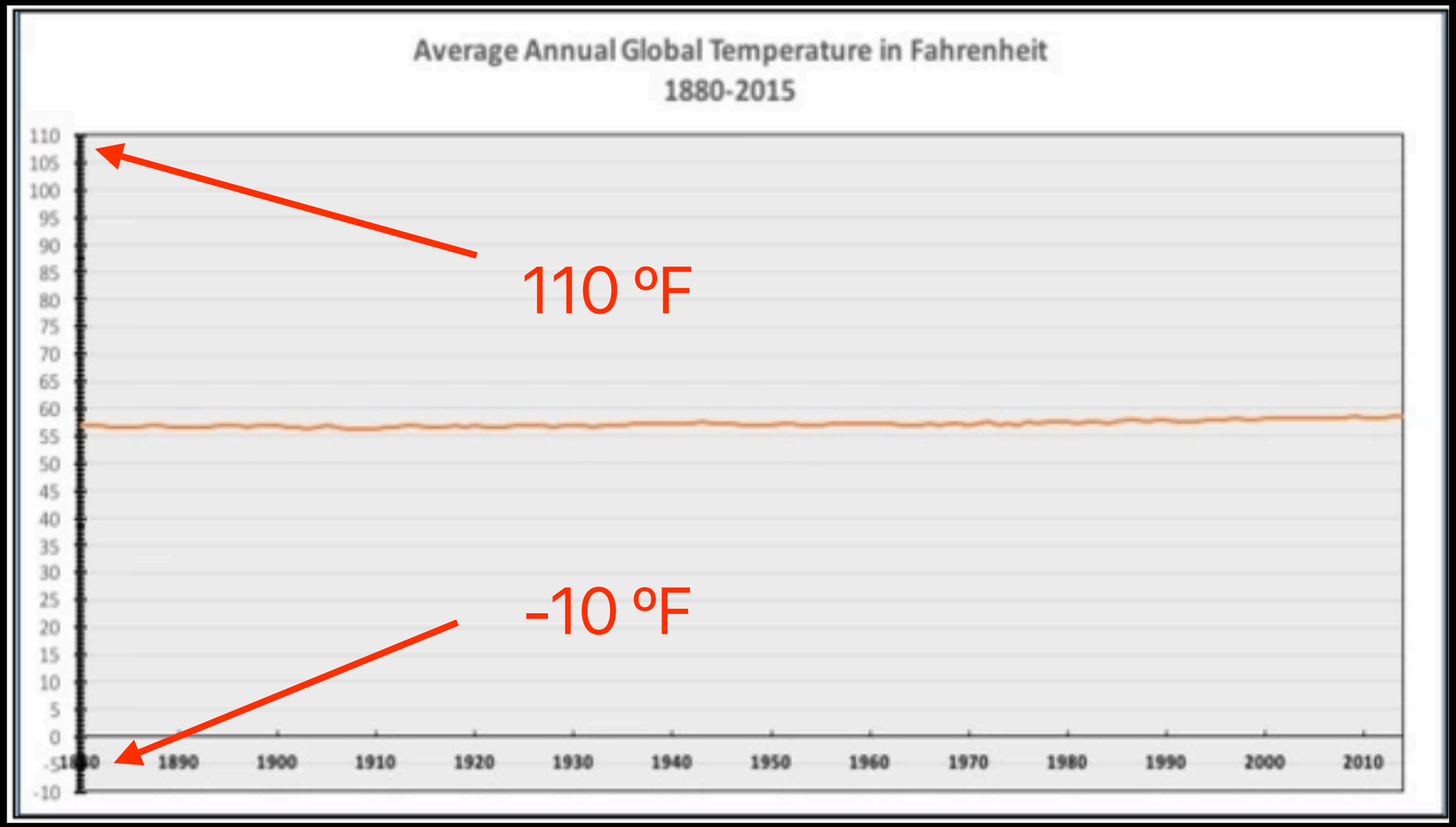
Week 3 – **Data Visualization, Tables**

Slides by Suraj Rampure

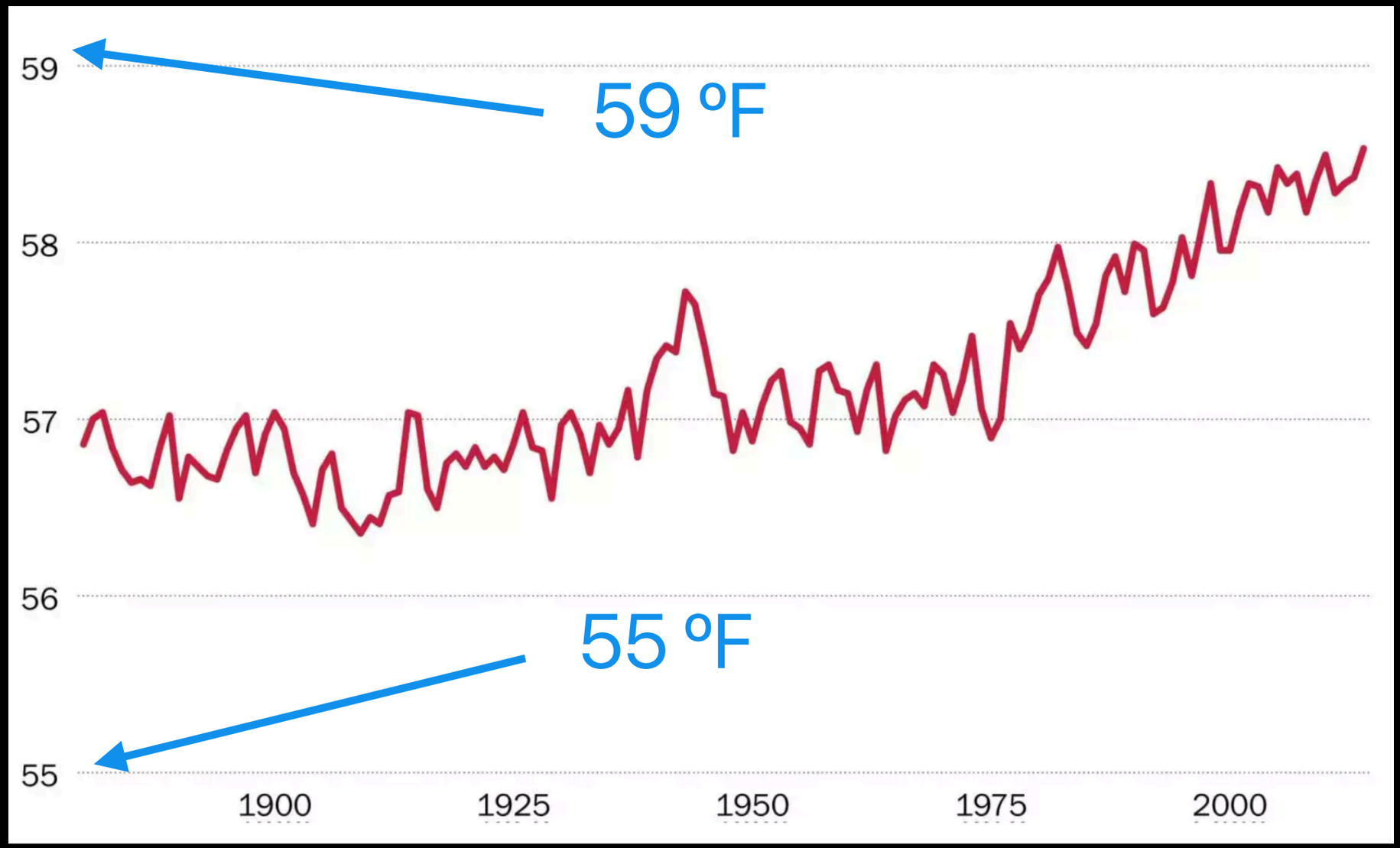
Data Visualization

Example: Global Temperature



Is global warming **#fakenews**?

Example: Global Temperature



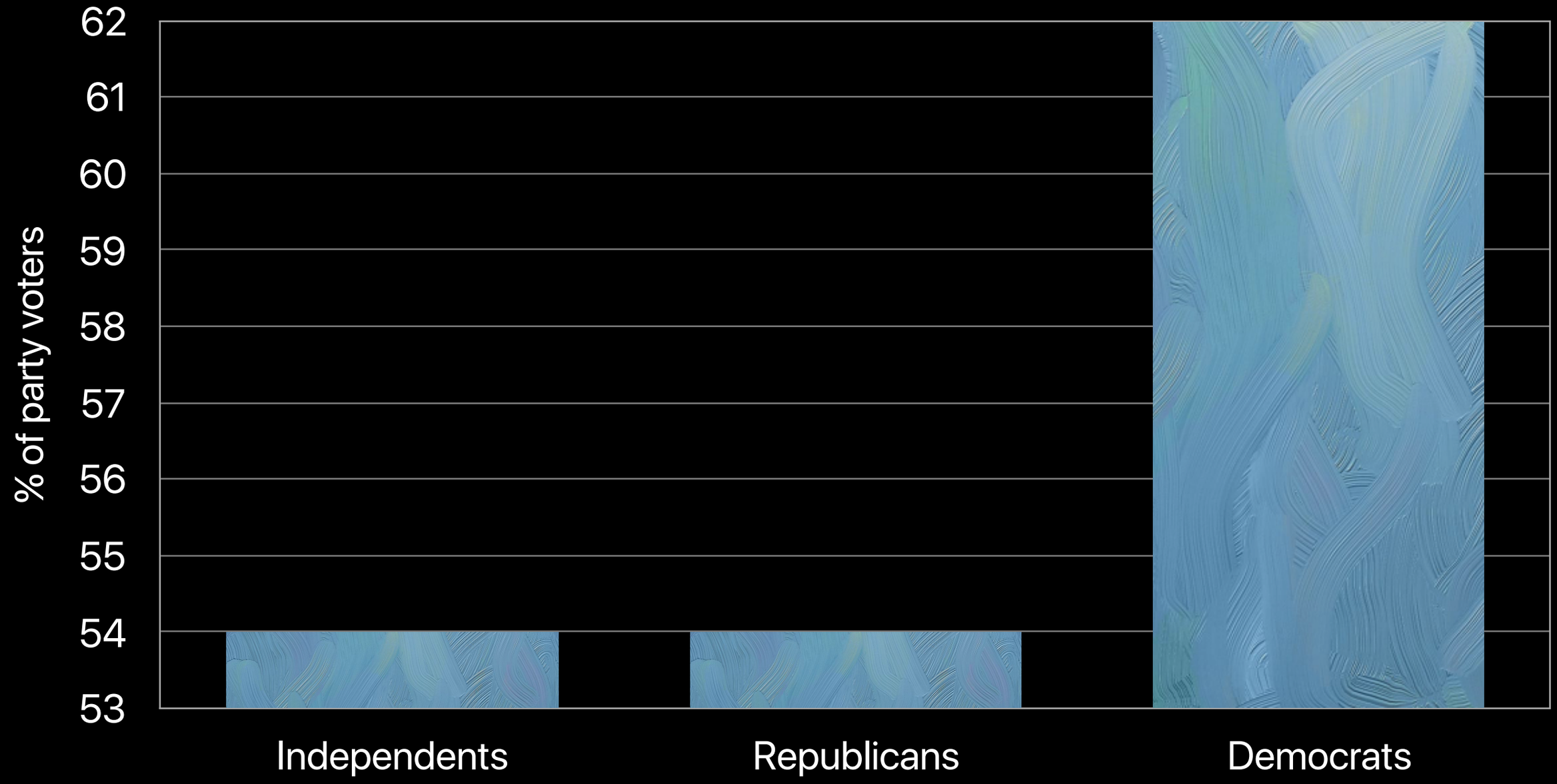
You can make data say **whatever you want it to**.

Try It Yourself – Party Affiliations

Party Affiliations	% That Agreed With Court Decision
Democrats	62
Republicans	54
Independents	54

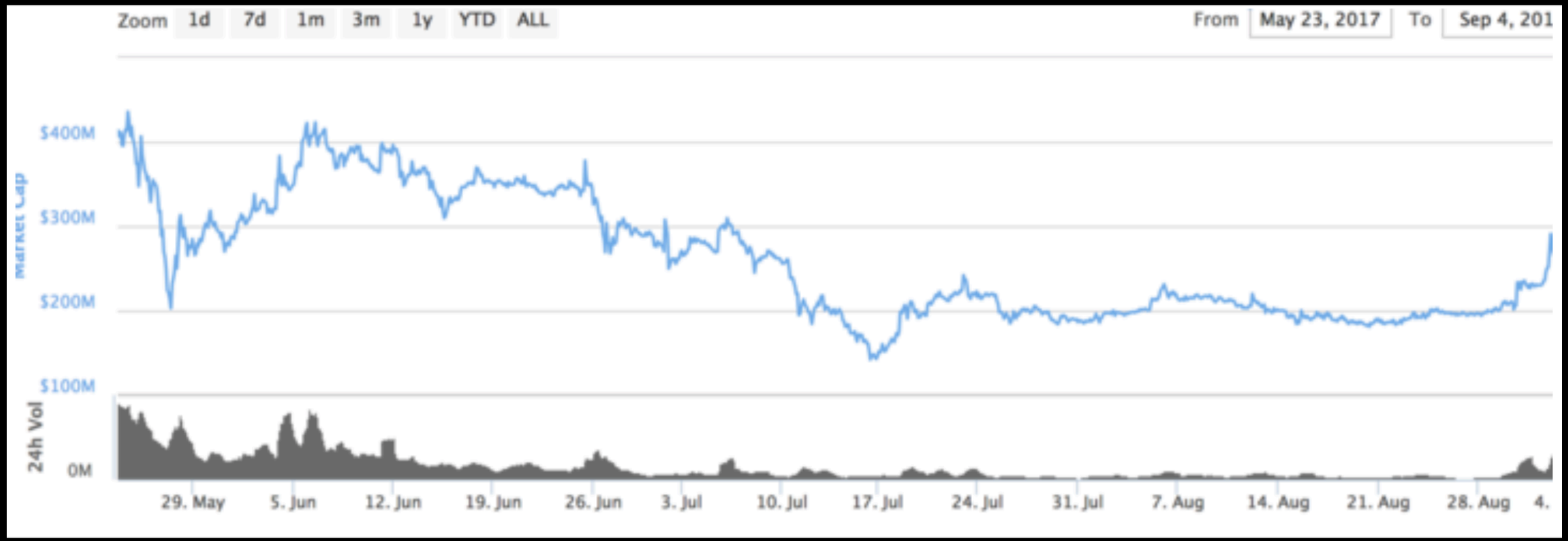
Can you find a way to plot this data in order to **make it appear as if** many more Democrats supported the ruling than Republicans or Independents?

Try It Yourself – Party Affiliations



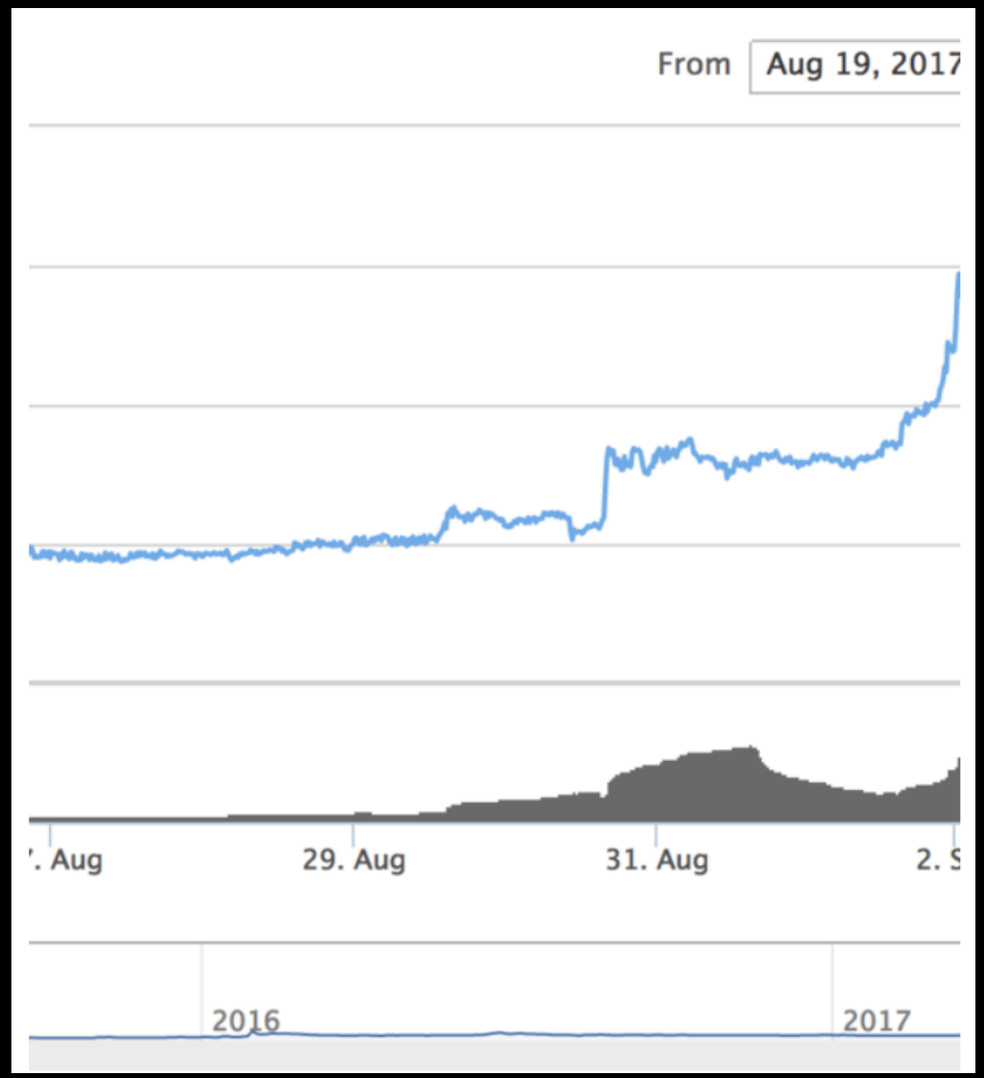
Scale is everything!

Try It Yourself – DogeCoin



Can you find a way to **make it look like** a great time to invest in DogeCoin?

Try It Yourself – DogeCoin



Some of the data may be excluded. **Watch out**!

Tables

Tables

Tables are a type of **two-dimensional dataset**, where columns represent different attributes. They belong to the **`datascience`** package.

`cars: Table`

company	average cost
Mercedes	55121
BMW	50324
Audi	53331
Porsche	65567
Volkswagen	35120

```
german_brands = make_array("Mercedes",
"BMW", "Audi", "Porsche", “Volkswagen")

average_costs = make_array(55121, 50324,
53331, 65567, 35120)

cars = Table().with_columns(
            "company", german_brands,
            "average cost", average_costs)
```

When creating a table, we call **`with_columns`** and alternate between column names and data.

Tables

`cars: Table`

company	average cost
Mercedes	55121
BMW	50324
Audi	53331
Porsche	65567
Volkswagen	35120

```
german_brands = make_array("Mercedes",
"BMW", "Audi", "Porsche", “Volkswagen")

average_costs = make_array(55121, 50324,
53331, 65567, 35120)

cars = Table().with_columns(
            "company", german_brands,
            "average cost", average_costs)
```

Each column is an array – this makes sense, since columns can only have one kind of value (think of Excel).

`cars.column(“company”)` or **`cars.column(0)`**

Tables

cars: Table

company	average cost
Mercedes	55121
BMW	50324
Audi	53331
Porsche	65567
Volkswagen	35120

```
german_brands = make_array("Mercedes",
"BMW", "Audi", "Porsche", “Volkswagen")

average_costs = make_array(55121, 50324,
53331, 65567, 35120)

cars = Table().with_columns(
            "company", german_brands,
            "average cost", average_costs)
```

Each column is an array – this makes sense, since columns can only have one kind of value (think of Excel).

`cars.column(“average cost”)` or `cars.column(1)`

Tables

cars: Table

company	average cost
Mercedes	55121
BMW	50324
Audi	53331
Porsche	65567
Volkswagen	35120

```
german_brands = make_array("Mercedes",
"BMW", "Audi", "Porsche", “Volkswagen")

average_costs = make_array(55121, 50324,
53331, 65567, 35120)

cars = Table().with_columns(
            "company", german_brands,
            "average cost", average_costs)
```

What if we want to add a new row?

Lists

```
stuff = [3, “hello”, Table(), 15.44, “this is weird”]
```

A list is a collection of ordered data, similar to an array. The main difference is that lists don’t need to contain data of all of the same type.

Tables

`cars: Table`

company	average cost
Mercedes	55121
BMW	50324
Audi	53331
Porsche	65567
Volkswagen	35120
Bugatti	1342331

```
german_brands = make_array("Mercedes",
"BMW", "Audi", "Porsche", “Volkswagen")

average_costs = make_array(55121, 50324,
53331, 65567, 35120)

cars = Table().with_columns(
            "company", german_brands,
            "average cost", average_costs)

new_brand = [“Bugatti”, 1342331]
cars = cars.with_row(new_brand)
```

`with_row` and **`with_rows`** are our friends.

If **`new_brand`** had more than two elements, an error would occur.

DEMO