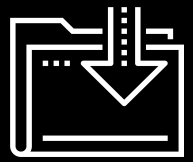
# Exploring Pandas

Data Boot Camp

Lesson 4.2



# Class Objectives

By the end of today's class, you will be able to:

Navigate through DataFrames using `loc` and `iloc`.

Filter and slice Pandas DataFrames.

Create and access Pandas `groupby` objects.

Sort DataFrames.

# Instructor Demonstration

## Exploring Data with loc and iloc

Programmers can easily collect specific rows and columns of data from a DataFrame by using the `loc` and `iloc` methods.

# Exploring Data with `loc` and `iloc`

- `loc` returns data based on an index of labels/strings
- `loc` is limited to string types and cannot be used on a numerical index. As an alternative solution, you can use the `df.set_index()` function, passing in the desired column header for the index.
- Instead of using labels, `iloc` uses integer-based indexing for selection by position.

```
In [4]: # Set new index to STREET NAME
        df = original_df.set_index("STREET NAME")
        df.head()
```

Out[4]:

| STREET NAME | STREET NAME ID | STREET FULL NAME | POSTAL COMMUNITY | MUNICIPAL COMMUNITY |
|---|---|---|---|---|
| **PRIVATE STREET** | 1400342 | PRIVATE STREET | BATON ROUGE | BATON ROUGE |
| **4TH** | 1 | N 4TH ST | BATON ROUGE | BATON ROUGE |
| **11TH** | 10 | S 11TH ST | BATON ROUGE | BATON ROUGE |
| **ADDINGTON** | 100 | ADDINGTON AVE | BATON ROUGE | BATON ROUGE |
| **CHALFONT** | 1000 | W CHALFONT DR | BATON ROUGE | PARISH |

# Exploring Data with `loc` and `iloc`

- Both `loc` and `iloc` use brackets that contain the desired rows, followed by a comma and the desired columns.
- For example, `loc[“ADDINGTON”, “STREET FULL NAME”]` or `iloc[3,1]`

In [5]:
```
# Grab the data contained within the "ADDINGTON" row and the "STREET FULL NAME" column
addington_name = df.loc["ADDINGTON", "STREET FULL NAME"]
print("Using Loc: " + addington_name)

also_addington_name = df.iloc[3, 1]
print("Using Iloc: " + also_addington_name)
```

```
Using Loc: ADDINGTON AVE
Using Iloc: ADDINGTON AVE
```

# Exploring Data with `loc` and `iloc`

Both methods allow us to select a range of columns and rows by providing a list.

We can also use a colon to tell Pandas to look for a range.

In [6]:
```
# Grab the first five rows of data and the columns from "STREET NAME ID" to "POSTAL COMMUNITY"
# The problem with using "STREET NAME" as the index is that the values are not unique so duplicates are returned
# If there are duplicates and loc[] is being used, Pandas will return an error
private_to_chalfont = df.loc[["PRIVATE STREET", "4TH", "11TH", "ADDINGTON",
                              "CHALFONT"], ["STREET NAME ID", "STREET FULL NAME", "POSTAL COMMUNITY"]]
print(private_to_chalfont)

print()

# Using iloc[] will not find duplicates since a numeric index is always unique
also_private_to_chalfont = df.iloc[0:5, 0:3]
print(also_private_to_chalfont)
```

```
                STREET NAME ID STREET FULL NAME POSTAL COMMUNITY
STREET NAME
PRIVATE STREET         1400342   PRIVATE STREET      BATON ROUGE
PRIVATE STREET         1400001   PRIVATE STREET      BATON ROUGE
PRIVATE STREET         1400015   PRIVATE STREET      BATON ROUGE
PRIVATE STREET         1400161   PRIVATE STREET      BATON ROUGE
PRIVATE STREET         1400343   PRIVATE STREET      BATON ROUGE
...                        ...             ...              ...
11TH                         9        N 11TH ST      BATON ROUGE
ADDINGTON                  100    ADDINGTON AVE      BATON ROUGE
CHALFONT                  1000    W CHALFONT DR      BATON ROUGE
CHALFONT                   998    N CHALFONT DR      BATON ROUGE
CHALFONT                   999    S CHALFONT DR      BATON ROUGE

[329 rows x 3 columns]

                STREET NAME ID STREET FULL NAME POSTAL COMMUNITY
STREET NAME
PRIVATE STREET         1400342   PRIVATE STREET      BATON ROUGE
4TH                          1         N 4TH ST      BATON ROUGE
11TH                        10        S 11TH ST      BATON ROUGE
ADDINGTON                  100    ADDINGTON AVE      BATON ROUGE
CHALFONT                  1000    W CHALFONT DR      BATON ROUGE
```

# Exploring Data with `loc` and `iloc`

By passing in a colon by itself, `loc` and `iloc` will select all rows or columns depending on where the colon is placed in relation to the comma.

In [7]:
```
# The following will select all rows for columns `STREET FULL NAME` and `POSTAL COMMUNITY`
df.loc[:, ["STREET FULL NAME", "POSTAL COMMUNITY"]].head()
```

Out[7]:

| STREET NAME | STREET FULL NAME | POSTAL COMMUNITY |
|---|---|---|
| **PRIVATE STREET** | PRIVATE STREET | BATON ROUGE |
| **4TH** | N 4TH ST | BATON ROUGE |
| **11TH** | S 11TH ST | BATON ROUGE |
| **ADDINGTON** | ADDINGTON AVE | BATON ROUGE |
| **CHALFONT** | W CHALFONT DR | BATON ROUGE |

`loc` and `iloc` can be used to conditionally filter rows of data based on the values within a column.

# Exploring Data with `loc` and `iloc`

- Instead of passing a list of indexes, we can use a logic statement.
- If multiple conditions should be checked, `&` and `|` may also be added into the logic test as representations of `and` and `or`.

In [9]:
```
# Loc and Iloc also allow for conditional statments to filter rows of data
# using Loc on the logic test above only returns rows where the result is True
only_prairieville = df.loc[df["POSTAL COMMUNITY"] == "PRAIRIEVILLE", :]
print(only_prairieville)

print()

# Multiple conditions can be set to narrow down or widen the filter
only_prairieville_and_jackson = df.loc[(df["POSTAL COMMUNITY"] == "PRAIRIEVILLE") | (
    df["POSTAL COMMUNITY"] == "JACKSON"), :]
print(only_prairieville_and_jackson)
```

```
                STREET NAME ID    STREET FULL NAME POSTAL COMMUNITY  \
STREET NAME
ALLIGATOR BAYOU          16497  ALLIGATOR BAYOU RD     PRAIRIEVILLE
BLUFF                    16498           BLUFF RD     PRAIRIEVILLE

             MUNICIPAL COMMUNITY
STREET NAME
ALLIGATOR BAYOU            PARISH
BLUFF                      PARISH

                STREET NAME ID      STREET FULL NAME POSTAL COMMUNITY  \
STREET NAME
TALMADGE                  4772          TALMADGE DR          JACKSON
TREAKLE                   4911           TREAKLE DR          JACKSON
DENNIS                    1452            DENNIS CT          JACKSON
ALLIGATOR BAYOU          16497    ALLIGATOR BAYOU RD     PRAIRIEVILLE
BLUFF                    16498           BLUFF RD     PRAIRIEVILLE
RENEE                     4072           RENEE CT          JACKSON
SANDY SPRINGS             4320     SANDY SPRINGS LN          JACKSON
SHANE                     4405           SHANE CT          JACKSON
BICKHAM                    518          BICKHAM RD          JACKSON
ADAMS                     5527           ADAMS LN          JACKSON
LA 68                     5838          LA 68 HWY          JACKSON
SIMMONS                   6105          SIMMONS LN          JACKSON
```

# Activity: Good Movies

In this activity, you will create an application that searches through IMDb data to find only the best movies out there.

Suggested Time:

20 minutes

# Activity: Good Movies

## Instructions:

- Use Pandas to load and display the CSV provided in `Resources`.
- List all the columns in the dataset.
- We're only interested in IMDb data, so create a new table that takes the film and all the columns related to IMDb.
- Filter out only the good movies—any film with an IMDb score greater than or equal to 7—and remove the norm ratings.
- Find less popular movies that you may not have heard about—anything with under 20,000 votes.
- Finally, export this file to a spreadsheet, excluding the index, so we can keep track of our future watchlist.

# Time’s Up! Let’s Review.

# Instructor Demonstration

## Cleaning Data

# Activity: Pandas Recap and Data Types

| | |
|---|---|
| **Instructions** | Open `PandasRecap.ipynb` in the `Unsolved` folder in your Jupyter notebook. |
| | Go through the cells, and follow the comments. |
| **Hints** | A list of a DataFrame's data types can be checked by accessing its `dtypes` property. |
| | To change a non-numeric column to a numeric column, use the `df.astype(<datatype>)` method and pass in the desired data type as the parameter. |

When dealing with massive datasets, it’s almost inevitable that we’ll encounter duplicate rows, inconsistent spelling, and missing values.

# Cleaning Data

`del <DataFrame>[<columns>]`

In [4]:
```
# Preview of the DataFrame
# Note that Memo_CD is likely a meaningless column
df.head()
```

Out[4]:

| | Name | Employer | City | State | Zip | Amount | Memo_CD |
|---|---|---|---|---|---|---|---|
| 0 | CAREY, JAMES | NOT EMPLOYED | HOCKESSIN | DE | 197071618.0 | 500 | NaN |
| 1 | OBICI, SILVANA | STONY BROOK | PORT JEFFERSON STATION | NY | 117764286.0 | 250 | NaN |
| 2 | MAISLIN, KAREN | RETIRED | WILLIAMSVILLE | NY | 14221.0 | 250 | NaN |
| 3 | MCCLELLAND, CARTER AND STEPHANIE | UNION SQUARE ADVISORS | NEW YORK | NY | 10023.0 | 1000 | NaN |
| 4 | MCCLUSKEY, MARTHA | STATE UNIVERSITY OF NEW YORK | BUFFALO | NY | 14214.0 | 250 | NaN |

In [5]:
```
# Delete extraneous column
del df['Memo_CD']
df.head()
```

Out[5]:

| | Name | Employer | City | State | Zip | Amount |
|---|---|---|---|---|---|---|
| 0 | CAREY, JAMES | NOT EMPLOYED | HOCKESSIN | DE | 197071618.0 | 500 |
| 1 | OBICI, SILVANA | STONY BROOK | PORT JEFFERSON STATION | NY | 117764286.0 | 250 |
| 2 | MAISLIN, KAREN | RETIRED | WILLIAMSVILLE | NY | 14221.0 | 250 |
| 3 | MCCLELLAND, CARTER AND STEPHANIE | UNION SQUARE ADVISORS | NEW YORK | NY | 10023.0 | 1000 |
| 4 | MCCLUSKEY, MARTHA | STATE UNIVERSITY OF NEW YORK | BUFFALO | NY | 14214.0 | 250 |

# Cleaning Data

`count()`
`<DataFrame>.dropna(how='any')`

In [6]:
```
# Identify incomplete rows
df.count()
```

Out[6]:
```
Name        2000
Employer    1820
City        1999
State       1999
Zip         1996
Amount      2000
dtype: int64
```

In [7]:
```
# Drop all rows with missing information
df = df.dropna(how='any')
```

In [8]:
```
# Verify dropped rows
df.count()
```

Out[8]:
```
Name        1818
Employer    1818
City        1818
State       1818
Zip         1818
Amount      1818
dtype: int64
```

# Cleaning Data

`value_counts()`
`replace()`

In [12]:
```
# Display an overview of the Employers column
df['Employer'].value_counts()
```

Out[12]:
```
NOT EMPLOYED           609
NONE                   321
SELF-EMPLOYED          132
SELF                    33
RETIRED                 32
                      ...
INTEL CORPORATION        1
SLOCUM & SONS            1
OCPS                     1
HEALTHCARE PARTNERS      1
CARBON FIVE              1
Name: Employer, Length: 519, dtype: int64
```

In [13]:
```
# Clean up Employer category. Replace 'SELF' and 'SELF EMPLOYED' with 'SELF-EMPLOYED'
df['Employer'] = df['Employer'].replace({'SELF': 'SELF-EMPLOYED', 'SELF EMPLOYED': 'SELF-EMPLOYED'})
```

In [14]:
```
# Verify clean-up.
df['Employer'].value_counts()
```

Out[14]:
```
NOT EMPLOYED           609
NONE                   321
SELF-EMPLOYED          180
RETIRED                 32
INGRAM BARGE COMPANY    30
```

# Activity: Portland Crime

In this activity, you will take a crime dataset from Portland, OR, and do your best to clean it up so that the DataFrame is consistent and no rows with missing data are present.

Suggested Time:

20 minutes

# Activity: Portland Crime

## Instructions:

Read in the CSV by using Pandas, and print out the DataFrame that is returned.

Get a count of the rows within the DataFrame to determine if there are any null values.

Drop the rows that contain null values.

Search through the `"Offense Type"` column, and replace any similar values with one consistent value.

Create a couple DataFrames that look into one neighborhood only, and print them to the screen.

# Time’s Up! Let’s Review.

# Activity: Pandas Recap and Data Types

In this activity, we will recap what has been covered in Pandas up to this point.

Suggested Time:

15 minutes


Countdown timer

15:00

(with alarm)

# Instructor Demonstration

## Pandas Grouping

`.groupby()` is a simpler method for filtering data.

# Pandas Grouping

To split the DataFrame into multiple groups and group by state, we use `df.groupby([<Columns>])`.

The `.groupby()` method returns a `groupby` object that can only be accessed by using a data function on it.

```
# Count how many loss incidents occured in each city
grouped_city_df = loss_df.groupby(["Incident City"])

print(grouped_city_df)

grouped_city_df.count().head(10)
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fd919ddadf0>
```

| Incident City | Fire Department Name | Incident date | Incident Type Code | Incident Type | Alarm Date and Time | Arrival Date and Time | Last Unit Cleared Date and Time | Property Loss | Contents Loss | Fire Service Deaths | Fire Service Injuries | Other Fire Deaths | Other Fire Injuries | Incident Zip Code | Response Time (seconds) | In Du (se |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AMSTON | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |
| ANSONIA | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | |
| AVON | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | |
| Andover | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | |
| Ansonia | 18 | 18 | 18 | 18 | 18 | 18 | 18 | 18 | 18 | 18 | 18 | 18 | 18 | 18 | 18 | |
| BERLIN | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | |
| BETHEL | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | |
| BLOOMFIELD | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | |
| BRANFORD | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | |
| BRIDGEPORT | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | |

# Pandas Grouping

The `pd.DataFrame()` method makes it possible to create new DataFrames by using only `groupby` data.

A DataFrame can also be created by selecting a single Series from a `groupby` object and passing it in as the values for a specified column.

```
# Save loss sums as series
city_property_loss = grouped_city_df["Property Loss"].sum()
city_contents_loss = grouped_city_df["Contents Loss"].sum()
city_contents_loss.head()
```

```
Incident City
AMSTON       5000.0
ANSONIA       600.0
AVON         1250.0
Andover       500.0
Ansonia    265100.0
Name: Contents Loss, dtype: float64
```

```
# Create a new DataFrame using count and loss amounts
city_summary_df = pd.DataFrame({"Number of Loss Incidents": city_loss_counts,
                                "Total Property Loss": city_property_loss,
                               "Total Contents Loss": city_contents_loss})
city_summary_df.head()
```

| | Number of Loss Incidents | Total Property Loss | Total Contents Loss |
|---|---|---|---|
| **AMSTON** | 1 | 65000.0 | 5000.0 |
| **ANSONIA** | 2 | 5000.0 | 600.0 |
| **AVON** | 6 | 14200.0 | 1250.0 |
| **Andover** | 3 | 2500.0 | 500.0 |
| **Ansonia** | 18 | 644100.0 | 265100.0 |

# Pandas Grouping

It's also possible to perform a `df.groupby()` method on multiple columns by passing two or more column references into the list parameter.

```
# It is also possible to group a DataFrame by multiple columns
# This returns an object with multiple indexes, however, which can be harder to deal with
grouped_city_loss_incidents = loss_df.groupby(["Incident City","Incident Type Code"])

grouped_city_loss_incidents.count().head(10)
```

| Incident City | Incident Type Code | Fire Department Name | Incident date | Incident Type | Alarm Date and Time | Arrival Date and Time | Last Unit Cleared Date and Time | Property Loss | Contents Loss | Fire Service Deaths | Fire Service Injuries | Other Fire Deaths | Other Fire Injuries |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AMSTON | 111 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ANSONIA | 111 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| AVON | 111 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| | 113 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 114 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 131 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Andover | 111 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| | 113 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Ansonia | 111 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 |
| | 113 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |

# Pandas Grouping

A new DataFrame can be created from a `groupby` object.

```
# Converting a GroupBy object into a DataFrame
total_city_loss_df = pd.DataFrame(
    grouped_city_loss_incidents[["Property Loss", "Contents Loss"]].sum())
total_city_loss_df.head(10)
```

| | | Property Loss | Contents Loss |
|---|---|---|---|
| **Incident City** | **Incident Type Code** | | |
| **AMSTON** | **111** | 65000.0 | 5000.0 |
| **ANSONIA** | **111** | 5000.0 | 600.0 |
| **AVON** | **111** | 8000.0 | 1200.0 |
| | **113** | 0.0 | 50.0 |
| | **114** | 1000.0 | 0.0 |
| | **131** | 5200.0 | 0.0 |
| **Andover** | **111** | 2500.0 | 200.0 |
| | **113** | 0.0 | 300.0 |
| **Ansonia** | **111** | 617100.0 | 263500.0 |
| | **113** | 5000.0 | 500.0 |

# Activity: Exploring U.S. Census Data

In this activity, you will revisit the U.S. Census data and create DataFrames with calculated totals and averages of each state by year.

Suggested Time:

25 minutes

# Activity: Exploring U.S. Census Data

**Instructions**

Read in the Census CSV file with Pandas.

Create two new DataFrames, one to find totals, and another to find averages. DataFrames should include:

- Totals for population, employed civilians, unemployed civilians, people in the military, and poverty count.
- Averages for median age, household income, and per capita income.

Create new DataFrames once these have been grouped by each year and state.

Rename any columns to reflect the data calculations.

Export the resulting tables to CSVs. We will use them again in our next class.

# Time’s Up! Let’s Review.

# Instructor Demonstration

## Sorting Made Easy

# Sorting Made Easy

To sort a DataFrame based on the values within a column, use the `df.sort_values()` method and pass in the column name to sort by as a parameter.

The "ascending" parameter is always marked as True by default. Therefore, the `sort_values()` method will always sort from lowest to highest unless the parameter of `ascending=False` is also passed into the `sort_values()` method.

```
# Sorting the DataFrame based on "Meals" column
# Will sort from lowest to highest if no other parameter is passed
meals_taxes_df = taxes_df.sort_values("Meals")
meals_taxes_df.head()
```

| | Town | Meals | Meals Count | Rent | Rent Count | Alcohol | Alcohol Count | Past Meals | Past Meals count | Past Rent | Past Rent Count | Past Alcohol | Past Alchohol Count |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | ADDISON | 0.0 | 0 | 90173.10 | 12 | 0.0 | 0 | 0.00 | 0 | 172233.00 | 15 | 0.0 | 0 |
| **98** | WELLS | 0.0 | 0 | 0.00 | 0 | 0.0 | 0 | 0.00 | 0 | 145041.00 | 11 | 0.0 | 0 |
| **35** | FAIRLEE | 0.0 | 0 | 1833212.02 | 10 | 0.0 | 0 | 2379763.68 | 11 | 4475959.53 | 12 | 0.0 | 0 |
| **36** | FAYSTON | 0.0 | 0 | 105586.77 | 11 | 0.0 | 0 | 0.00 | 0 | 211939.30 | 19 | 0.0 | 0 |
| **37** | FERRISBURGH | 0.0 | 0 | 0.00 | 0 | 0.0 | 0 | 7025450.58 | 11 | 5829011.70 | 15 | 0.0 | 0 |

```
# To sort from highest to lowest, ascending=False must be passed in
meals_taxes_df = taxes_df.sort_values("Meals", ascending=False)
meals_taxes_df.head()
```

| | Town | Meals | Meals Count | Rent | Rent Count | Alcohol | Alcohol Count | Past Meals | Past Meals count | Past Rent | Past Rent Count | Past Alcohol | Past Alchohol Count |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **17** | BURLINGTON | 74507552.54 | 219 | 18230026.80 | 26 | 18324508.20 | 122 | 1.276183e+08 | 236 | 53634054.09 | 44 | 44233463.37 | 129 |
| **81** | SOUTH BURLINGTON | 64445667.13 | 111 | 13750969.61 | 19 | 4138460.85 | 40 | 8.953598e+07 | 117 | 38211751.51 | 25 | 10313786.70 | 44 |
| **77** | RUTLAND | 38005509.10 | 98 | 1508769.29 | 14 | 2973734.52 | 38 | 4.199332e+07 | 98 | 3822279.43 | 14 | 5316214.36 | 38 |
| **32** | ESSEX | 36429036.93 | 91 | 0.00 | 0 | 2359611.62 | 29 | 4.203358e+07 | 104 | 0.00 | 0 | 4129281.23 | 31 |
| **12** | BRATTLEBORO | 33966669.55 | 102 | 4868408.74 | 26 | 2840765.10 | 41 | 4.144862e+07 | 100 | 9867296.43 | 27 | 6096085.57 | 42 |

# Activity: Search for the Worst

In this activity, you will take a dataset on San Francisco Airport's utility consumption and determine which day in the dataset had the worst consumption for each utility.

Suggested Time:

20 minutes

# Activity: Search For the Worst

## Instructions:

- Read in the CSV file provided, and print it to the screen.
- Print out a list of all the values within the "Utility" column.
- Select a value from this list, and create a new DataFrame that only includes that utility. Note that some utilities have more than one option for "Owner," and you may want to limit this new DataFrame to a single "Owner."
- Sort the DataFrame based on the level of consumption, from most to least.
- Reset the index for the DataFrame so that the index is in order.
- Print out the details of the worst day to the screen.

# Time’s Up! Let’s Review.

# Questions?