# Evaluating distribution choices

MONTE CARLO SIMULATIONS IN PYTHON

**Izzy Weber**
Curriculum Manager, DataCamp

# Choosing variable probability distributions

1. Gain an intuitive understanding of data and available probability distributions
2. Use Maximum Likelihood Estimation (MLE) to compare candidate distributions
3. Use Kolmogorov-Smirnov test to evaluate *goodness of fit* of probability distributions
   - Quantifies distance between the empirical distribution of the data and the theoretical candidate probability distribution
   - Use `scipy.stats.kstest()` to calculate

# Evaluating choice of distribution: age

```
results = []
list_of_dists = ["laplace", "norm", "expon"]
for i in list_of_dists:
    dist = getattr(st, i)
    param = dist.fit(dia["age"])
    result = st.kstest(dia["age"], i, args=param)
    print(result)
```

Results for Laplace, normal, and exponential distributions in that order:

```
KstestResult(statistic=0.09511179937112832, pvalue=0.0006239579389182981)
KstestResult(statistic=0.0615913626181368, pvalue=0.0670322523435981 1)
KstestResult(statistic=0.2536037941921312, pvalue=1.5202547969084796e-25)
```

# Evaluating choice of distribution: age

Results for Laplace, normal, and exponential distributions in that order:

```
KstestResult(statistic=0.09511179937112832, pvalue=0.0006239579389182981)
KstestResult(statistic=0.0615913626181368, pvalue=0.06703225234359811)
KstestResult(statistic=0.2536037941921312, pvalue=1.5202547969084796e-25)
```

# Evaluating choice of distribution: tc blood serum

```
results = []
list_of_dists = ["laplace", "norm", "expon"]
for i in list_of_dists:
    dist = getattr(st, i)
    param = dist.fit(dia["tc"])
    result = st.kstest(dia["tc"], i, args=param)
    print(result)
```

Results for Laplace, normal, and exponential distributions in that order:

```
KstestResult(statistic=0.06435779928393615, pvalue=0.04915329841106708)
KstestResult(statistic=0.05116529574722724, pvalue=0.19085587687385897)
KstestResult(statistic=0.3318461436889846, pvalue=7.018486943525e-44)
```

# Let's practice!

MONTE CARLO SIMULATIONS IN PYTHON

# Visualizing simulation results

MONTE CARLO SIMULATIONS IN PYTHON

**Izzy Weber**
Curriculum Manager, DataCamp

# Answering questions using Monte Carlo simulations

What are the differences in the predicted y values for people who are in the fourth quartile of each predictor compared to the first quartile?

# Starting with the simulated results

```
df_summary.head()
```

|   | age | bmi | bp | tc | ldl | hdl | tch | ltg | glu | predicted_y |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 54.491842 | 32.512362 | 82.131464 | 203.075420 | 114.043050 | 44.820017 | 5.137683 | 5.254633 | 100.815909 | 209.297688 |
| 1 | 66.380490 | 29.380708 | 98.810054 | 136.474760 | 68.457982 | 51.691298 | 3.455412 | 4.572478 | 96.117969 | 177.081339 |
| 2 | 59.003285 | 27.015225 | 92.195168 | 242.796424 | 126.541644 | 86.050629 | 2.423928 | 4.640063 | 87.485747 | 123.192425 |
| 3 | 34.803821 | 20.961365 | 86.852597 | 168.762268 | 110.113823 | 53.158621 | 3.925988 | 4.080205 | 79.187999 | 84.284908 |
| 4 | 56.732615 | 32.682115 | 118.384860 | 226.152964 | 136.838283 | 46.467736 | 4.376397 | 5.374001 | 104.184429 | 244.900141 |

# Answering our question

```python
dic_diffs = {}
for var in ["age", "bmi", "bp", "tc", "ldl", "hdl", "tch", "ltg", "glu"]:
    var_q25 = np.quantile(df_summary[var], 0.25)
    var_q75 = np.quantile(df_summary[var], 0.75)
    q75_outcome = np.mean(df_summary[(df_summary[var] > var_q75)]["predicted_y"])
    q25_outcome = np.mean(df_summary[(df_summary[var] < var_q25)]["predicted_y"])
    y_diff = q75_outcome - q25_outcome
    dic_diffs[var] = [y_diff]
df_diffs = pd.DataFrame.from_dict(dic_diffs)
df_diffs.head()
```

```
     age      bmi      bp      tc     ldl     hdl     tch      ltg     glu
0 36.721  114.462  87.101  42.625  35.413 -77.653  82.835  105.611  74.744
```

# Simulating 1,000 times

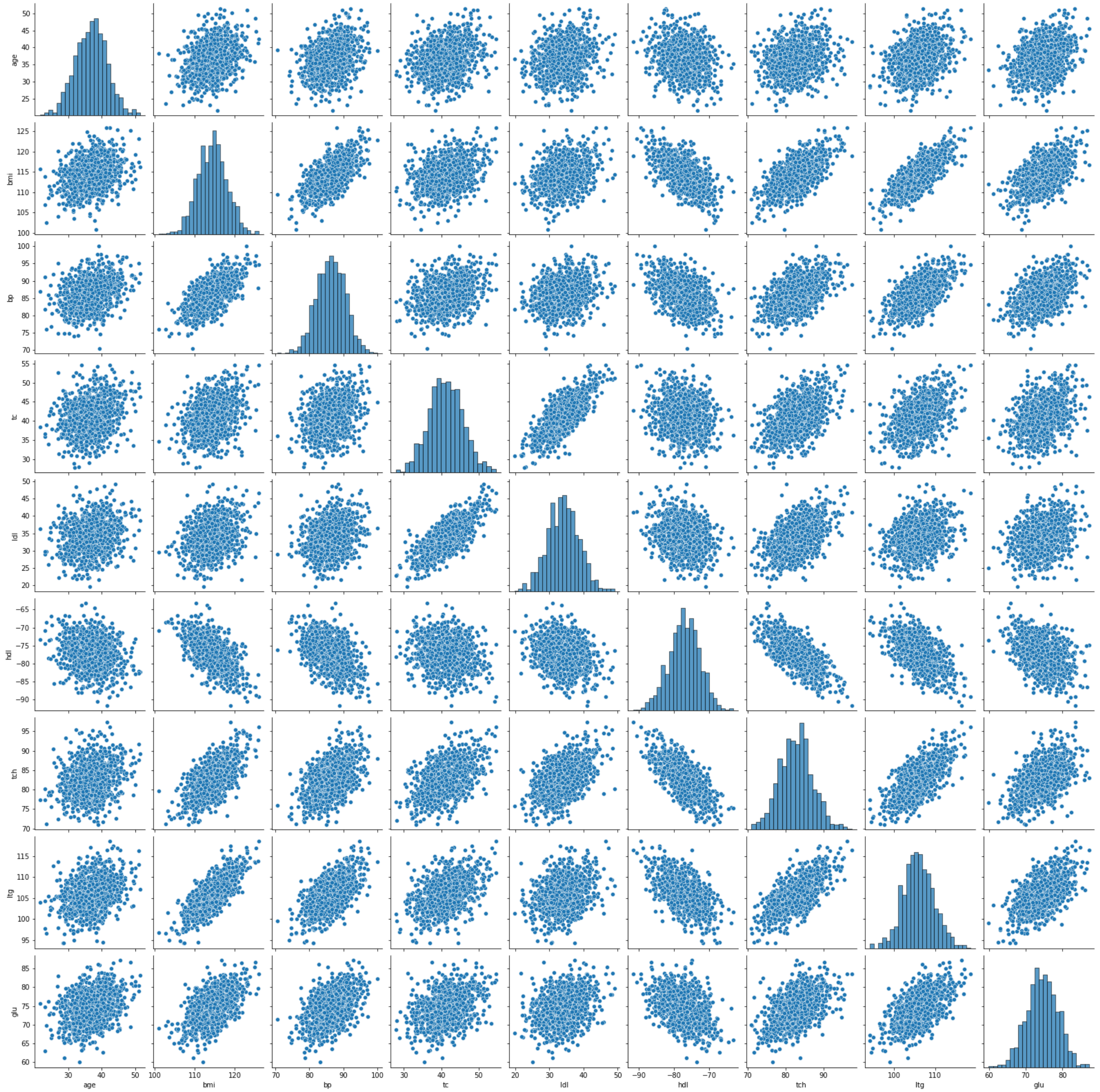
```
df_diffs.head()
```

```
        age      bmi       bp       tc      ldl      hdl      tch      ltg      glu
0    30.045  108.973  81.107  36.728  31.060  -75.885  79.829  101.231  67.215
1    36.860  112.486  86.164  39.596  33.970  -71.609  77.367  100.964  72.301
2    30.387  110.651  79.972  44.655  39.583  -74.377  82.354  103.808  71.804
3    35.047  113.609  83.241  34.706  23.764  -75.798  83.070  102.119  75.230
4    31.050  109.022  81.727  41.590  31.136  -73.955  80.866  101.668  74.225
```

- This DataFrame has 1,000 rows: one row for each set of mean difference from each of the 1,000 simulations

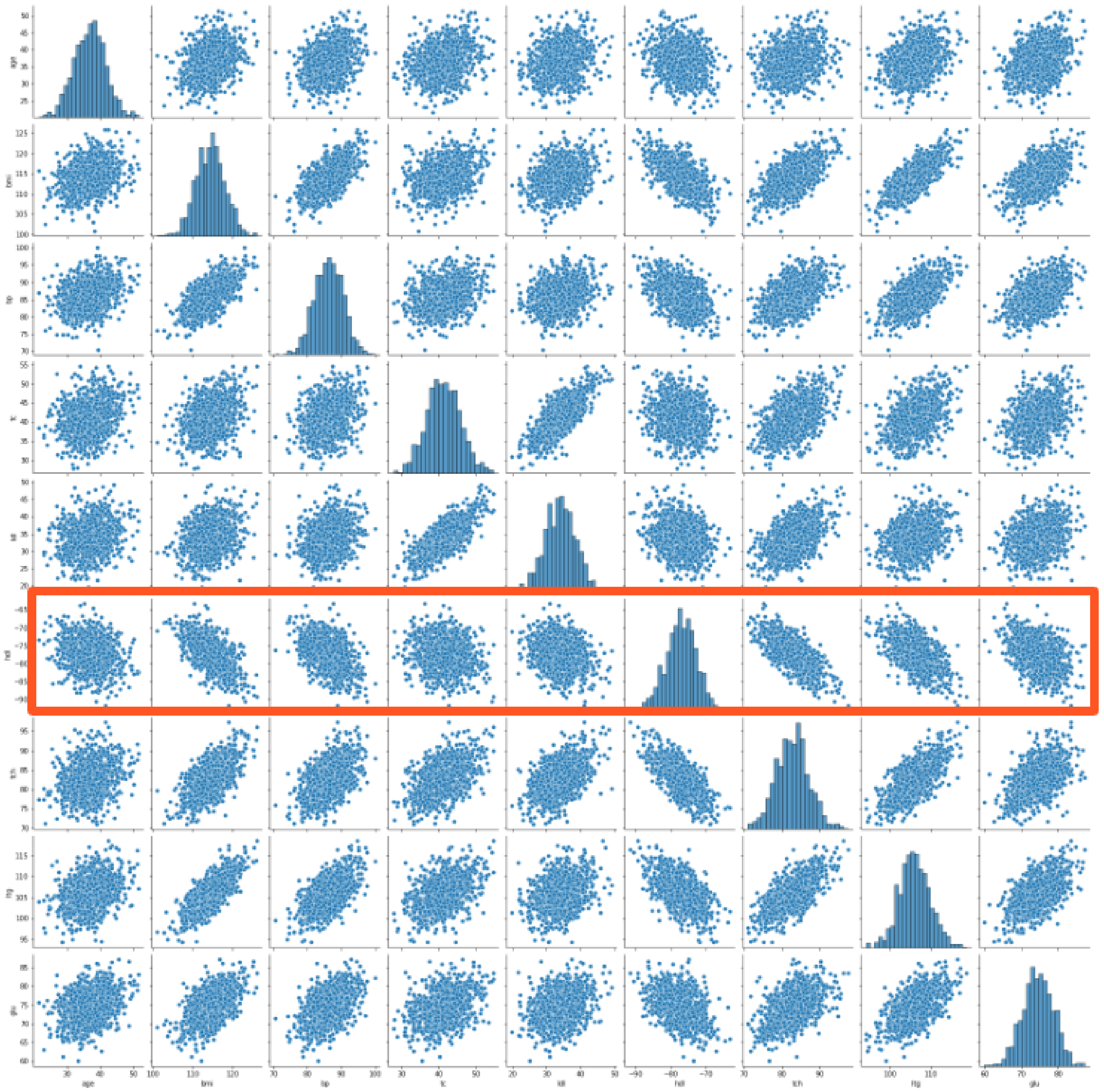
# Pairplot

```
sns.pairplot(df_diffs)
```



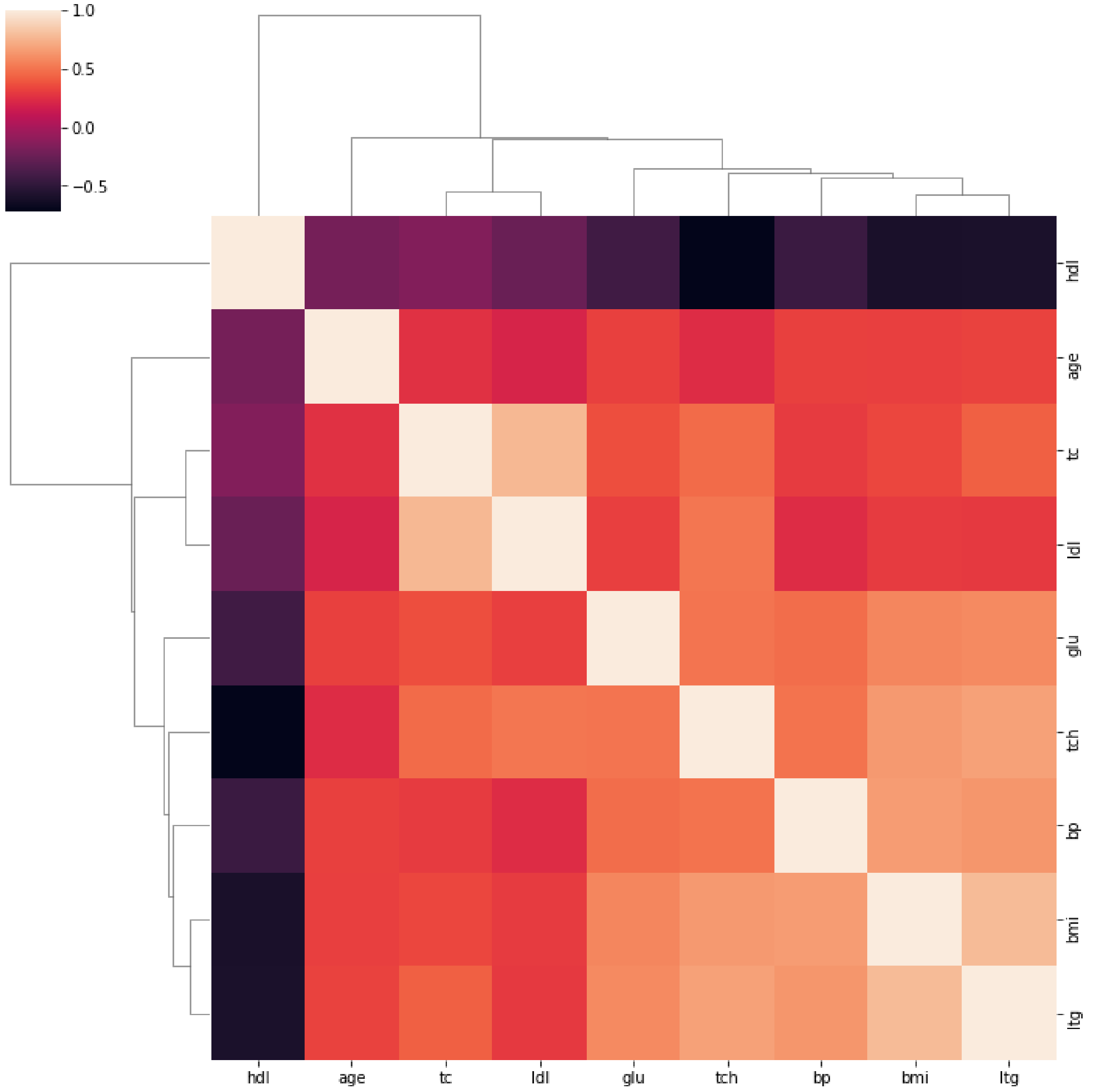
# Pairplot

```
sns.pairplot(df_diffs)
```



# Clustermap

```
sns.clustermap(df_diffs.corr())
```



# Converting to long format

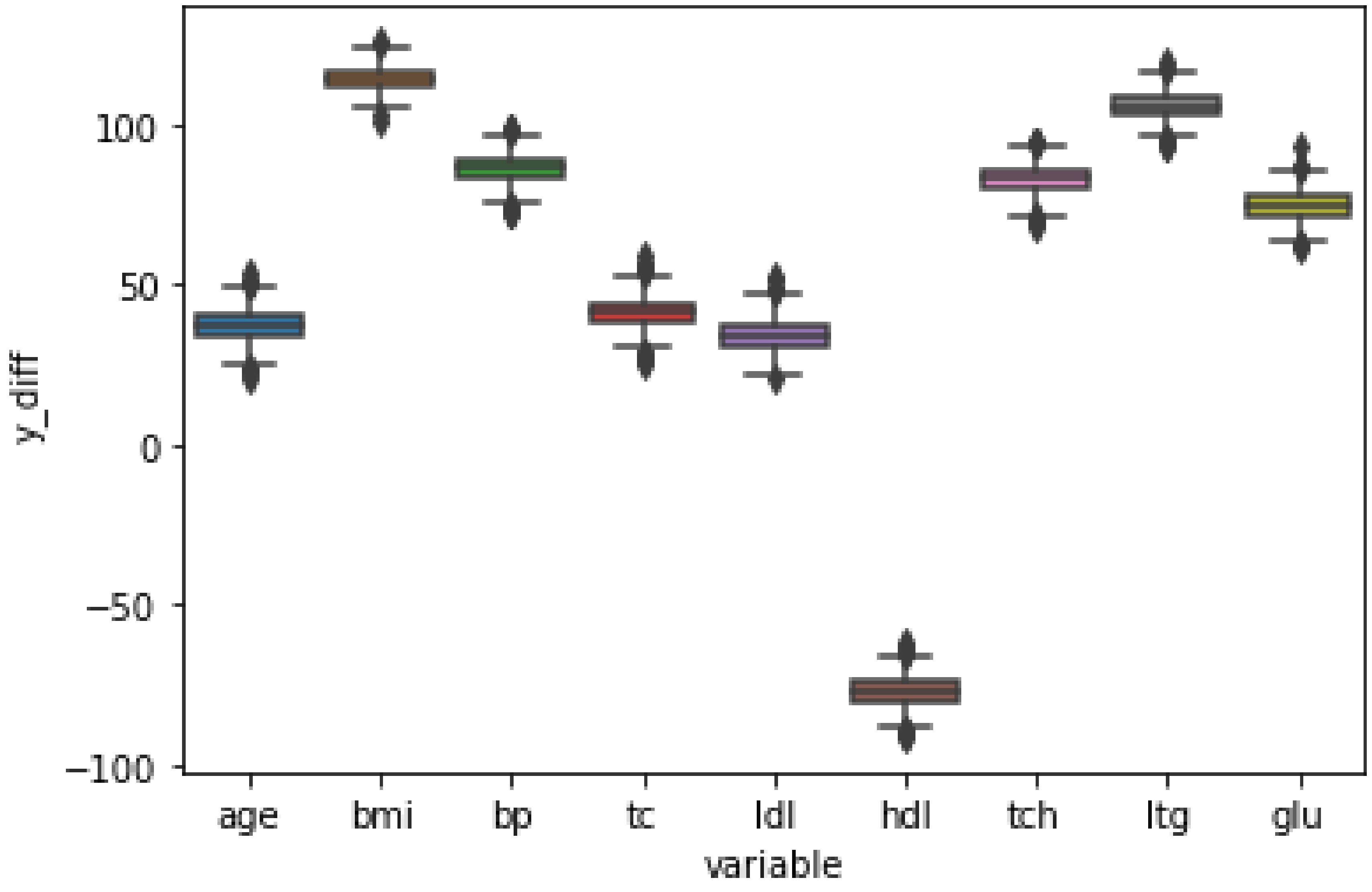
- DataFrames in long format often contain two columns: the variable name and the other the corresponding value

```
df_diffs_long=df_diffs.melt(value_name="y_diff",
                            value_vars=["age", "bmi", "bp", "tc", "ldl", "hdl",
                                        "tch", "ltg", "glu"])
df_diffs_long.head()
```

```
    variable        y_diff
0   age             30.045
1   age             36.860
2   age             30.387
3   age             35.047
4   age             31.050
```

# Boxplot

```
sns.boxplot(x="variable", y="y_diff", data=df_diffs_long)
```



# Let's practice!

MONTE CARLO SIMULATIONS IN PYTHON

# Sensitivity analysis

MONTE CARLO SIMULATIONS IN PYTHON

**Izzy Weber**
Curriculum Manager, DataCamp

# Sensitivity analysis

- Helps us understand the impact of the range of inputs
- Illustrates the patterns or trends when summarized in tables or plots

*If we increase or decrease the values for* `bmi` *and* `hdl` *using a Monte Carlo simulation, how will the predicted y values (disease progression) change?*

# Defining the parameters

```
cov_dia = dia[["age", "bmi", "bp", "tc", "ldl", "hdl", "tch", "ltg", "glu"]].cov()
mean_dia = dia[["age", "bmi", "bp", "tc", "ldl", "hdl", "tch", "ltg", "glu"]].mean()
```

# Defining the simulation function

```
def simulate_bmi_hdl(cov_dia, mean_list):
    list_ys = []
    for i in range(50):
        simulation_results = st.multivariate_normal.rvs(mean=mean_list,
                                                        size=500, cov=cov_dia)
        df_results = pd.DataFrame(simulation_results,
                                  columns=["age","bmi","bp","tc","ldl","hdl","tch","ltg","glu"])
        predicted_y = regr_model.predict(df_results)
        df_y = pd.DataFrame(predicted_y, columns=["predicted_y"])
        df_summary = pd.concat([df_results, df_y], axis=1)
        y = np.mean(df_summary["predicted_y"])
        list_ys.append(y)
    return(np.mean(list_ys))
```

# Perform simulations with a range of input parameters

```
hdl = []
bmi = []
simu_y = []
for mean_hdl_inc in np.arange(-20, 50, 30):
    for mean_bmi_inc in np.arange(-7, 11, 3):
        mean_list = mean_dia + np.array([0, mean_bmi_inc, 0, 0, 0, mean_hdl_inc, 0, 0, 0])
        hdl.append(mean_hdl_inc)
        bmi.append(mean_bmi_inc)
        mean_y = simulate_bmi_hdl(cov_dia, mean_list)
        simu_y.append(mean_y)
df_sa = pd.concat([pd.Series(hdl), pd.Series(bmi), pd.Series(simu_y)], axis=1)
df_sa.columns = ["hdl_inc", "bmi_inc", "y"]
```

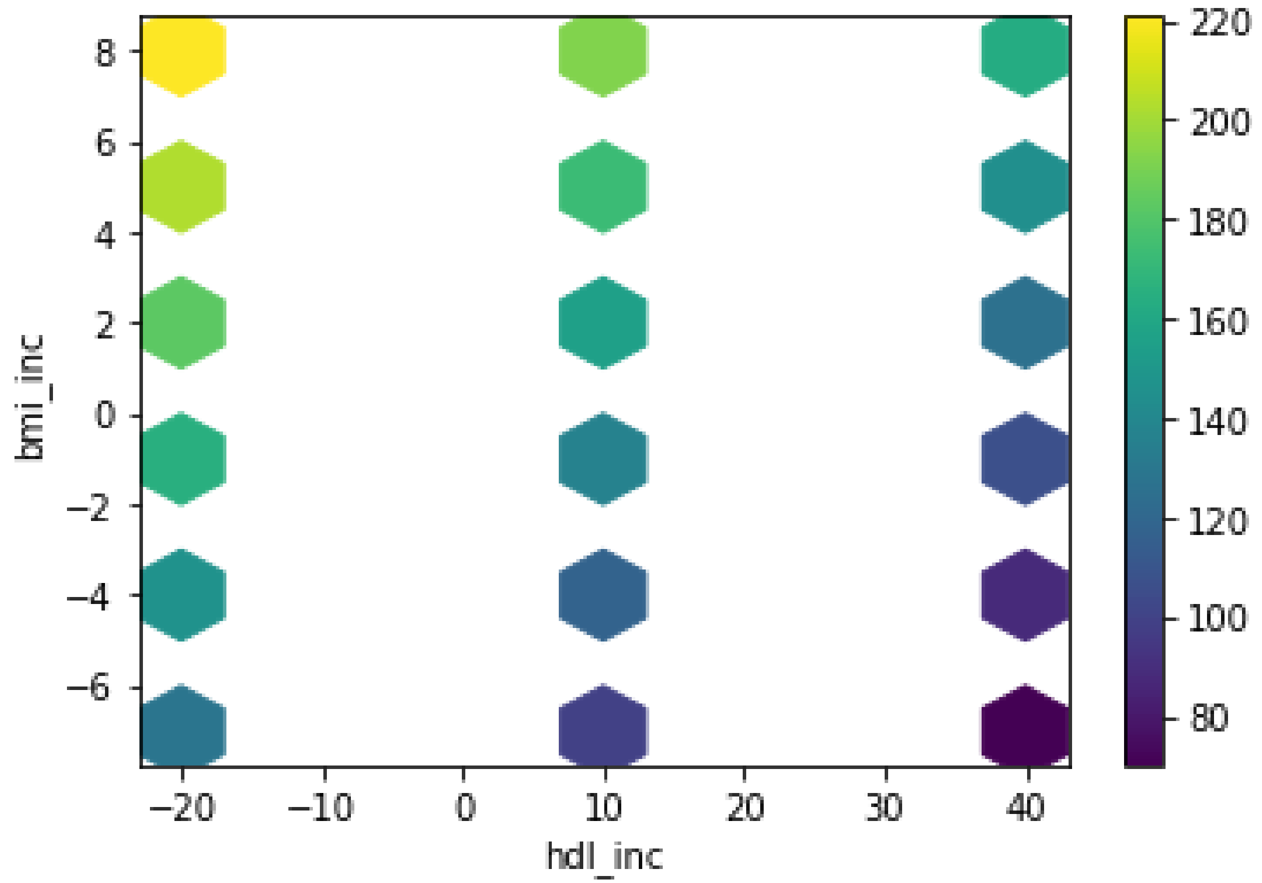
# Styled DataFrames of sensitivity analysis results

```
df_sa.sort_values(by=['hdl_inc', 'bmi_inc']).pivot(index='hdl_inc',
                                                   columns='bmi_inc',
                                                   values='y').style.background_gradient(
                                                   cmap=sns.light_palette("red", as_cmap=True))
```

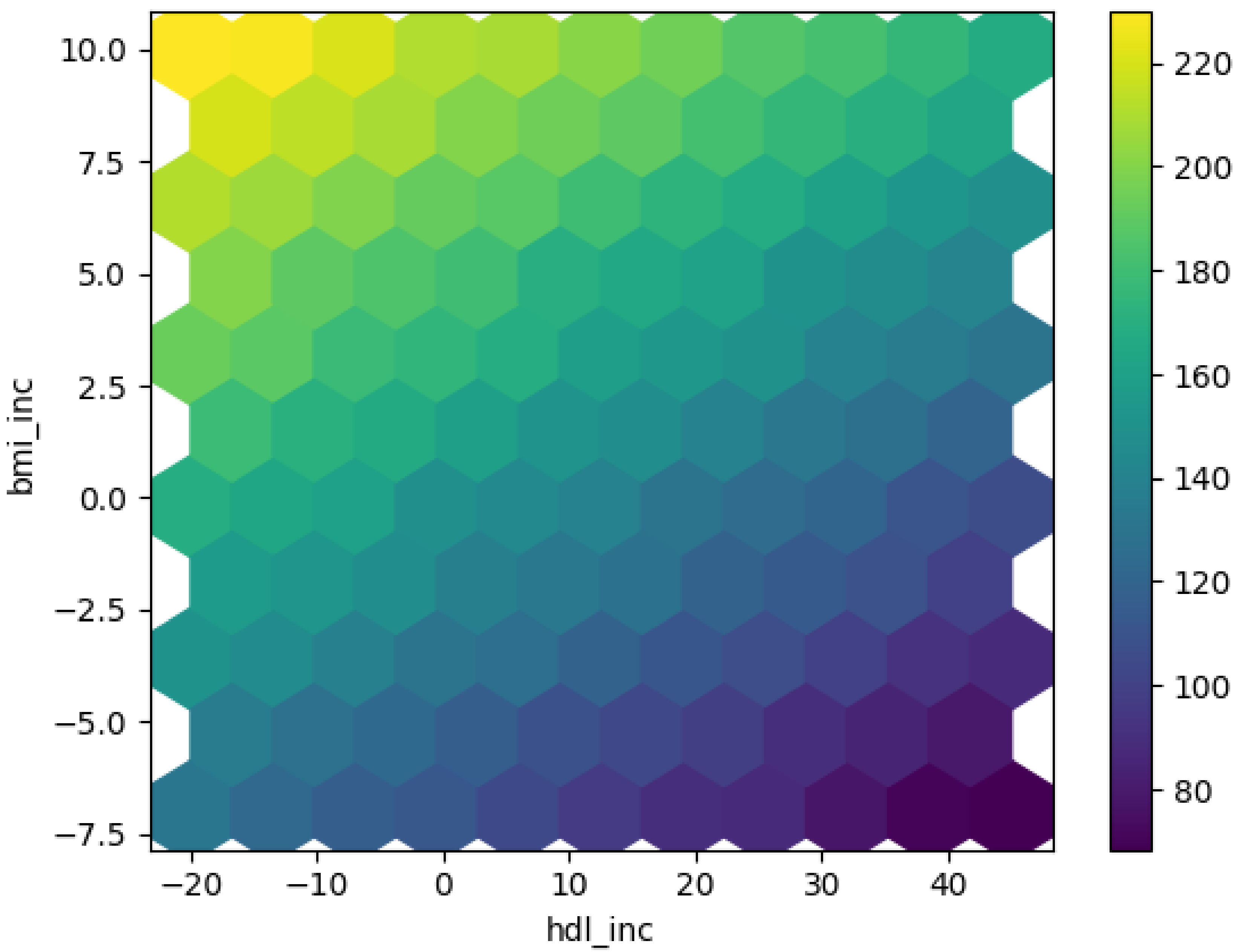
| bmi_inc<br>hdl_inc | -7 | -4 | -1 | 2 | 5 | 8 |
|---|---|---|---|---|---|---|
| -20 | 128.313347 | 146.788965 | 165.237888 | 183.925198 | 202.439964 | 220.860332 |
| 10 | 99.150789 | 117.993128 | 136.040590 | 154.108108 | 173.100882 | 191.778971 |
| 40 | 70.351470 | 89.323106 | 106.627512 | 125.809040 | 144.166632 | 162.095737 |

# Hexbin plot for sensitivity analysis results

```
df_sa.plot.hexbin(x='hdl_inc',y='bmi_inc', C='y',
                  reduce_C_function=np.mean,
                  gridsize=10, cmap="viridis",
                  sharex=False)
```



# Hexbin plot for dense parameter space



# Let's practice!

MONTE CARLO SIMULATIONS IN PYTHON

# Congratulations!

MONTE CARLO SIMULATIONS IN PYTHON

**Izzy Weber**
Curriculum Manager, DataCamp

# What we've covered

- What Monte Carlo simulations are and why they are useful
- Sampling from probability distributions as the basis for Monte Carlo simulations
- The steps of performing Monte Carlo simulations
- How to further evaluate and interpret simulation results

# Tips on further learning

1. **Continue learning about statistics**
   - DataCamp skill track: Statistics Fundamentals with Python
2. **Expand on your data visualization skills**
   - Introduction to Data Visualization with Seaborn
   - Intermediate Data Visualization with Seaborn

# Thank you!

MONTE CARLO SIMULATIONS IN PYTHON