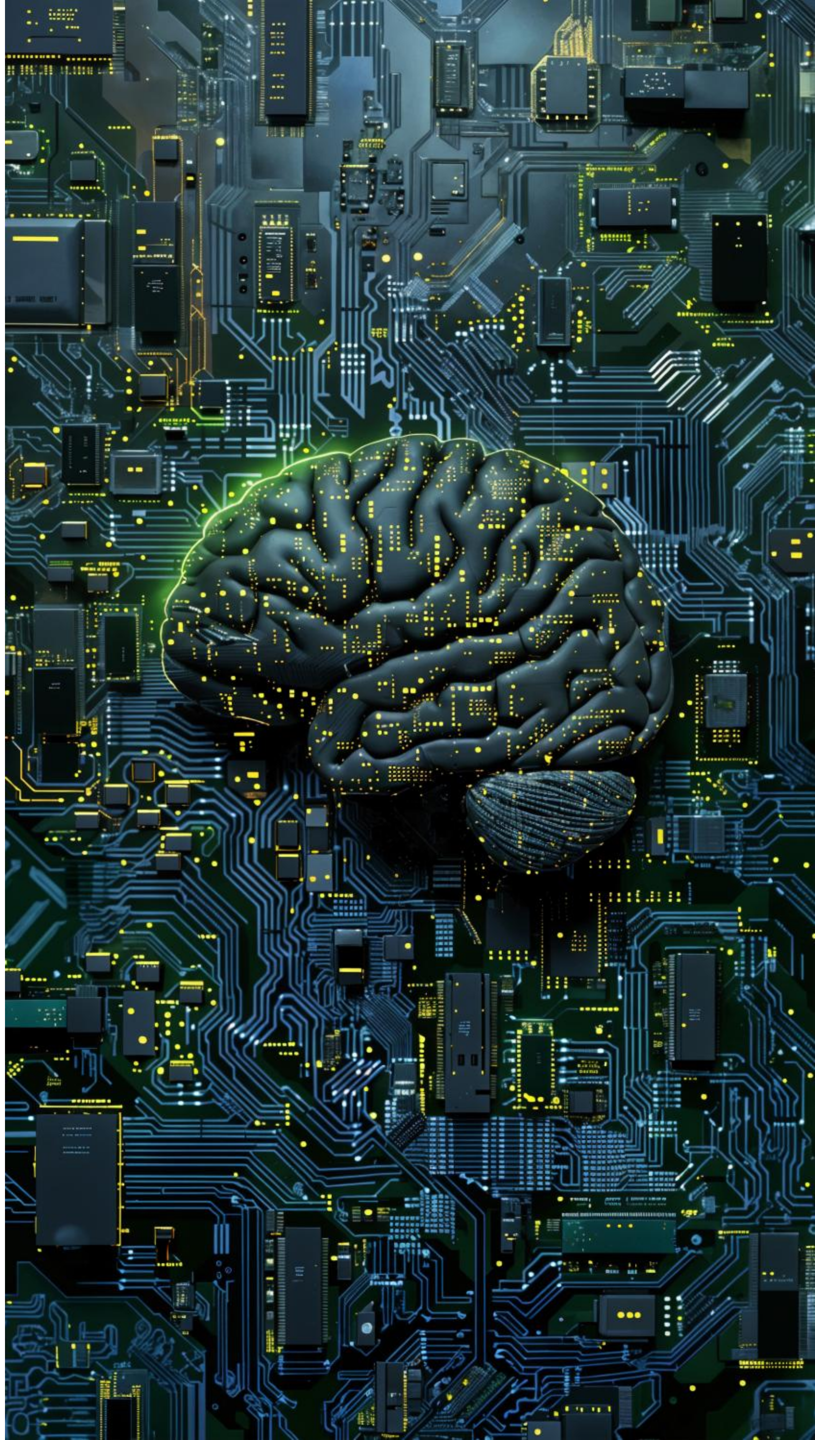
Machine Learning

Week 3 – Working with data, classification

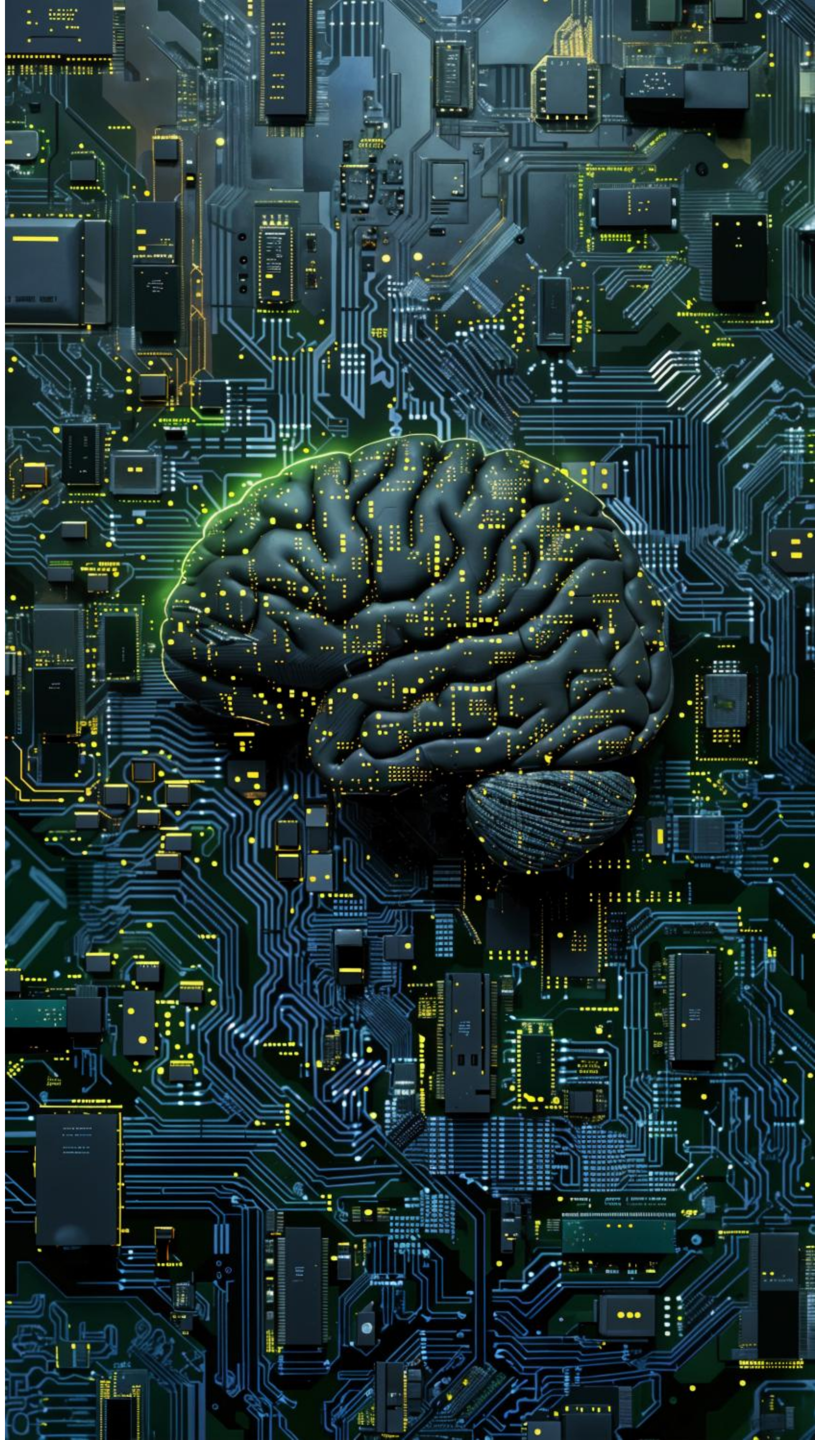
SAXION UNIVERSITY OF APPLIED SCIENCES



Working with data

Types of data, augmenting, filling, transforming

SAXION UNIVERSITY OF APPLIED SCIENCES



Pandas is your friend

Most of the data formats can be read by pandas:

```python
import pandas as pd

df = pd.read_csv("dataset.csv", sep=",")
```

Pandas works with `DataFrame`'s

Pandas is your friend

Looking at the data

```
data.head() #take a look at the data
```

feature

...	HSP	EA	EB	EC	SP
0	N	Below expectations	6.2	Missing value	21.0
1	E	Below expectations	Missing value	Missing value	Missing value
2	N	Below expectations	14.6	13.0	33.0
3	K	Missing value	Missing value	48.3	13.0
4	N	Below expectations	36.6	Missing value	3.0

observation

Pandas is your friend

Looking at the data

```
data.head() #take a look at the data
```

...	HSP	EA	EB	EC	SP
0	N	Below expectations	6.2	*Missing value*	21.0
1	E	Below expectations	*Missing value*	*Missing value*	*Missing value*
2	N	Below expectations	14.6	13.0	33.0
3	K	*Missing value*	*Missing value*	48.3	13.0
4	N	Below expectations	36.6	*Missing value*	3.0

index categorical features numerical features

Pandas is your friend

Selecting rows (observations) by **Row Index**

```
df.loc[4]
```

	4
HSP	N
EA	Below expectations
EB	36.6
EC	*Missing value*
SP	3.0

Pandas is your friend

Selecting columns

```
df.loc[:,"EA"]
df["EA"]
```

	EA
0	Below expectations
1	Below expectations
2	Below expectations
3	*Missing value*
4	Below expectations
5	Meets expectations
6	*Missing value*
7	Sufficient
8	Below expectations
9	Good

78 rows x 1 cols 10 per page

```
df.loc[:,["SP", "HSP"]]
df[["SP", "HSP"]]
```

	# SP	HSP
0	21.0	N
1	*Missing value*	E
2	33.0	N
3	13.0	K
4	3.0	N
5	24.0	E
6	45.0	G
7	28.0	E
8	46.0	K
9	11.0	K

78 rows x 2 cols 10 per page

Pandas is your friend

Combining row and column selection

```
df.loc[5:9, "EB":]
```

...	# EB	# EC	# SP
5	0.8	*Missing value*	24.0
6	62.4	*Missing value*	45.0
7	24.0	4.0	28.0
8	4.6	51.0	46.0
9	*Missing value*	7.0	11.0

Pandas is your friend

Row and column selection by 0-based (array-like) indexing

```
df.iloc[5:9, 2:]
```

...	# EB	# EC	# SP
5	0.8	*Missing value*	24.0
6	62.4	*Missing value*	45.0
7	24.0	4.0	28.0
8	4.6	51.0	46.0

Pandas is your friend

Selecting data based on condition

```
# select items where SP <= 40 and HSP taking values of N or E
df[(df["SP"] <= 40) & df['HSP'].isin(['N', 'E']) ]
```

...	HSP	EA	# EB	# EC	# SP
0	N	Below expectations	6.2	Missing value	21.0
2	N	Below expectations	14.6	13.0	33.0
4	N	Below expectations	36.6	Missing value	3.0
5	E	Meets expectations	0.8	Missing value	24.0
7	E	Sufficient	24.0	4.0	28.0

Pandas: basic cleaning

Looking at the data once more

```
data.head() #take a look at the data
```

...	HSP	EA	# EB	# EC	# SP
0	N	Below expectations	6.2	*Missing value*	21.0
1	E	Below expectations	*Missing value*	*Missing value*	*Missing value*
2	N	Below expectations	14.6	13.0	33.0
3	K	*Missing value*	*Missing value*	48.3	13.0
4	N	Below expectations	36.6	*Missing value*	3.0

Missing value
NaN (for numerical)
Null (for others)

Pandas: basic cleaning

How many missing values do we have?

```
print(data.isna().sum())
```

...	# 0
HSP	0
EA	3
EB	4
EC	9
SP	1

```
df = df.dropna()
# or in-place
df.dropna(inplace=True)
```

Just remove observations that have missing features

Pandas: basic cleaning

Based on domain knowledge (if you know what your data is):

- EA, EB, EC are exam grades
- SP are study points

replace missing values with 0's:

```
# replace NaNs in EB, EC, SP with 0
df[["EB", "EC", "SP"]] = df[["EB", "EC", "SP"]].fillna(0)

# replace NaNs in EA with "Below expectations"
df["EA"] = df["EA"].fillna("Below expectations")

# Or perform in-place replacement
df.fillna({"EA": "Below expectations"}, inplace=True)
```

It's better to use scikit-learn for this!

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="constant", fill_value=0)

df[:] = imputer.fit_transform(df[:])
```

	HSP	EA	EB	EC	SP
0	N	Below expectations	6.2	0.0	21.0
1	E	Below expectations	0.0	0.0	0.0
2	N	Below expectations	14.6	13.0	33.0
3	K	0	0.0	48.3	13.0
4	N	Below expectations	36.6	0.0	3.0

Strategies:

- constant
- mean
- median
- most_frequent

It’s better to use scikit-learn for this!

```
pipeline = Pipeline(steps=[

("imputer", ColumnTransformer(
  transformers=[
   ("numeric", SimpleImputer(strategy="constant", fill_value=0), ["EB", "EC", "SP"]),
   ("other", SimpleImputer(strategy="constant", fill_value="Below expectations"), ["EA"]),
  ],
  remainder="passthrough"
 )),

 ("scaler", MinMaxScaler()),

 ("regressor", LinearRegression()),
])
```

Pandas: knowing data

	HSP	EA	EB	EC	SP
0	N	Below expectations	6.2	0.0	21.0
1	E	Below expectations	0.0	0.0	0.0
2	N	Below expectations	14.6	13.0	33.0
3	K	Below expectations	0.0	48.3	13.0
4	N	Below expectations	36.6	0.0	3.0
5	E	Meets expectations	0.8	0.0	24.0
6	G	Below expectations	62.4	0.0	45.0
7	E	Sufficient	24.0	4.0	28.0
8	K	Below expectations	4.6	51.0	46.0
9	K	Good	0.0	7.0	11.0

Pandas: knowing data

Transforming categorical data:

- Ordered data: `OrdinalEncoder`
- Unordered data: `OneHotEncoder`
- Target variable: `LabelEncoder`

Pandas: knowing data

	HSP	EA	EB	EC	SP
0	N	Below expectations	6.2	0.0	21.0
1	E	Below expectations	0.0	0.0	0.0
2	N	Below expectations	14.6	13.0	33.0
3	K	Below expectations	0.0	48.3	13.0
4	N	Below expectations	36.6	0.0	3.0
5	E	Meets expectations	0.8	0.0	24.0
6	G	Below expectations	62.4	0.0	45.0
7	E	Sufficient	24.0	4.0	28.0
8	K	Below expectations	4.6	51.0	46.0
9	K	Good	0.0	7.0	11.0

EA is ordered

1. *"Below expectations"*
2. *"Sufficient"*
3. *"Meets expectations"*
4. *"Good"*
5. *"Exceeds expectations"*

HSP seems unordered

'N', 'E', 'K', 'G', 'B'

SAXION UNIVERSITY OF APPLIED SCIENCES

Encoding ordered categorical data

```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder(
      categories=[["Below expectations",
                   "Sufficient",
                   "Meets expectations",
                   "Good",
                   "Excellent"]])

df["EA_"] = ordinal_encoder.fit_transform(df[["EA"]])
```

	EA	EA_
0	Below expectations	0.0
1	Below expectations	0.0
2	Below expectations	0.0
3	Below expectations	0.0
4	Below expectations	0.0
5	Meets expectations	2.0
6	Below expectations	0.0
7	Sufficient	1.0
8	Below expectations	0.0
9	Good	3.0

Encoding **ordered** categorical data (pipeline variant)

```
pipeline = Pipeline(steps=[
  ("imputer", ColumnTransformer( ...)),
  ("ordinal_encoder", ColumnTransformer(
   transformers=[
     ("EA", OrdinalEncoder(categories=[["Below expectations",
                                        "Sufficient",
                                        "Meets expectations",
                                        "Good",
                                        "Excellent"]]), ["EA"]),
     ],
   remainder="passthrough"
  )),
  ...
])
```

Encoding **unordered** categorical data

```
from sklearn.preprocessing import OneHotEncoder


onehot_encoder = OneHotEncoder(sparse_output=False, drop="if_binary")

hsp_encoded = onehot_encoder.fit_transform(df[["HSP"]])

hsp_encoded_df = pd.DataFrame(hsp_encoded,
                              columns=onehot_encoder.get_feature_names_out(["HSP"]))

pd.concat([df["HSP"], hsp_encoded_df], axis=1).head()
```

Encoding unordered categorical data

```
from sklearn.preproce

onehot_encoder = OneH                                                    _binary")

hsp_encoded = onehot_

hsp_encoded_df = pd.D
                                                                         ture_names_out(["HSP"]))

pd.concat([df["HSP"],
```

	HSP	HSP_B	HSP_E	HSP_G	HSP_K	HSP_N
0	N	0.0	0.0	0.0	0.0	1.0
1	E	0.0	1.0	0.0	0.0	0.0
2	N	0.0	0.0	0.0	0.0	1.0
3	K	0.0	0.0	0.0	1.0	0.0
4	N	0.0	0.0	0.0	0.0	1.0
5	E	0.0	1.0	0.0	0.0	0.0
6	G	0.0	0.0	1.0	0.0	0.0
7	E	0.0	1.0	0.0	0.0	0.0
8	K	0.0	0.0	0.0	1.0	0.0
9	K	0.0	0.0	0.0	1.0	0.0

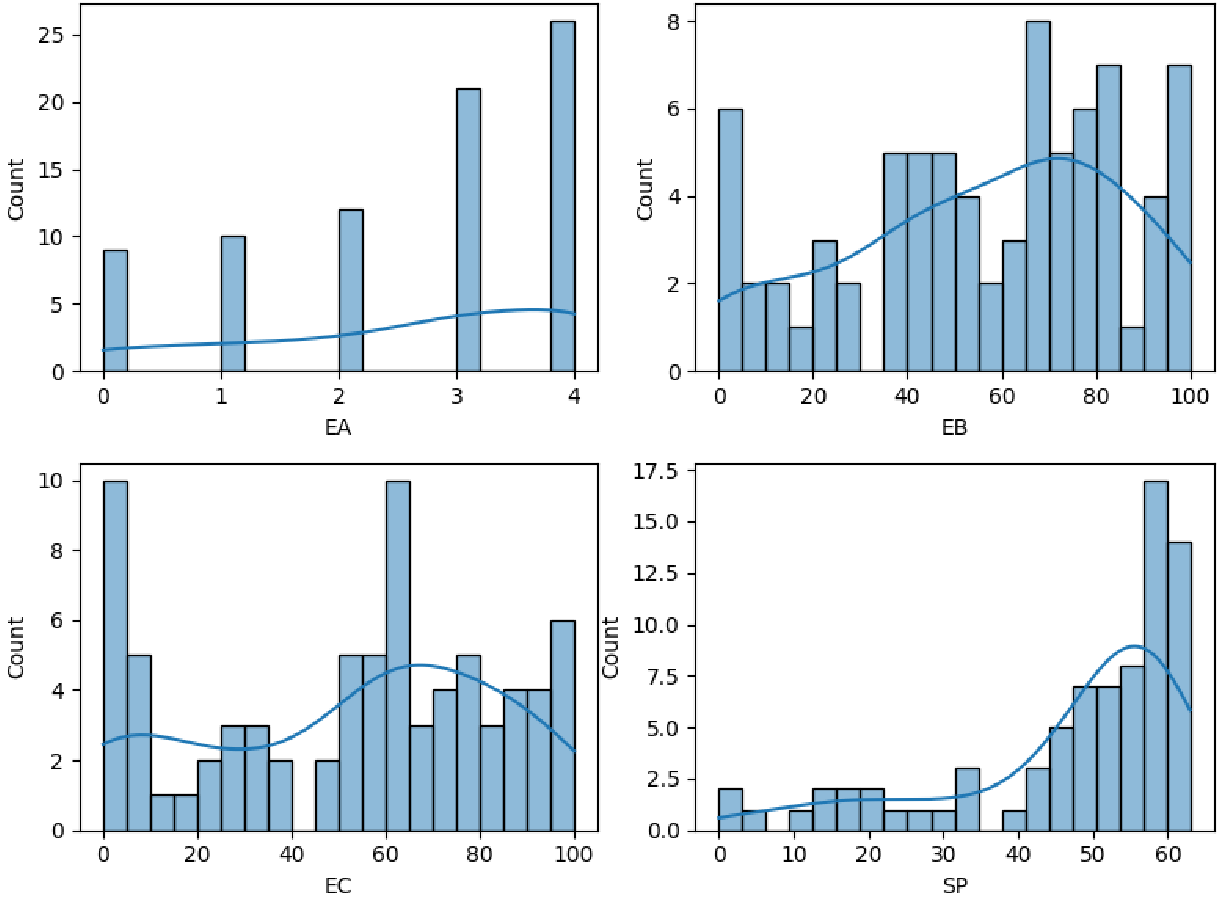
Encoding **unordered** categorical data (pipeline variant)

```
pipeline = Pipeline(steps=[
    ("imputer", ColumnTransformer( ...)),
    ("ordinal_encoder", ColumnTransformer( ...)),


    ("onehot_encoder", ColumnTransformer(
        transformers=[
            ("HSP", OneHotEncoder(sparse_output=True, drop="if_binary"), ["HSP"]),
        ],
        remainder="passthrough"
    )),
    ...
])
```

Take a deeper look at the data: distributions

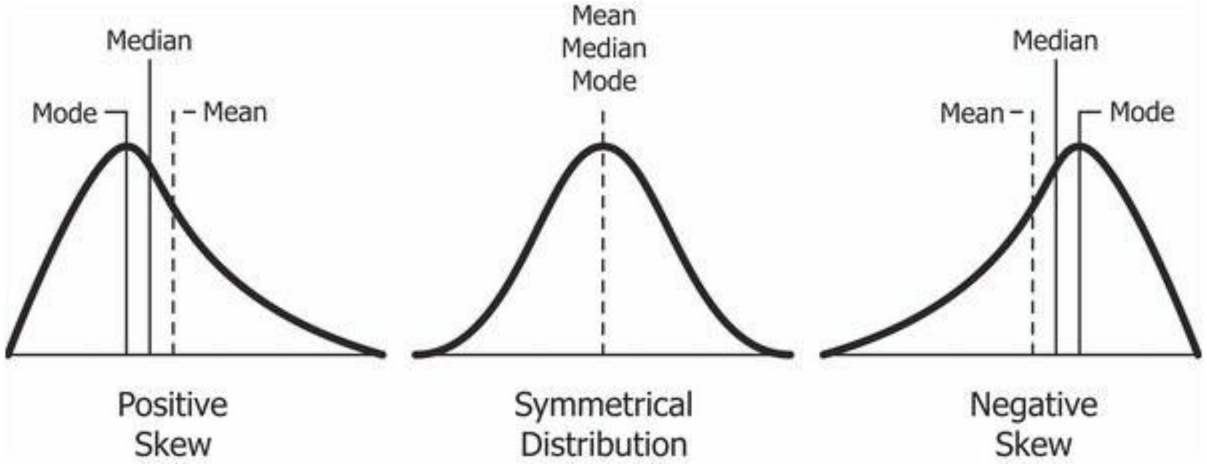
```
import seaborn as sns
import matplotlib.pyplot as plt


# plot histograms only for EA, EB, EC, SP
fig, axs = plt.subplots(ncols=2, nrows=2, figsize=(8, 6))
index = 0
axs = axs.flatten()
for k, v in df[['EA', 'EB', 'EC', 'SP']].items():
    sns.histplot(v, ax=axs[index], bins=20, bins=True)
    index += 1
plt.tight_layout()
```

Take a deeper look at the data: distributions



SAXION UNIVERSITY OF APPLIED SCIENCES

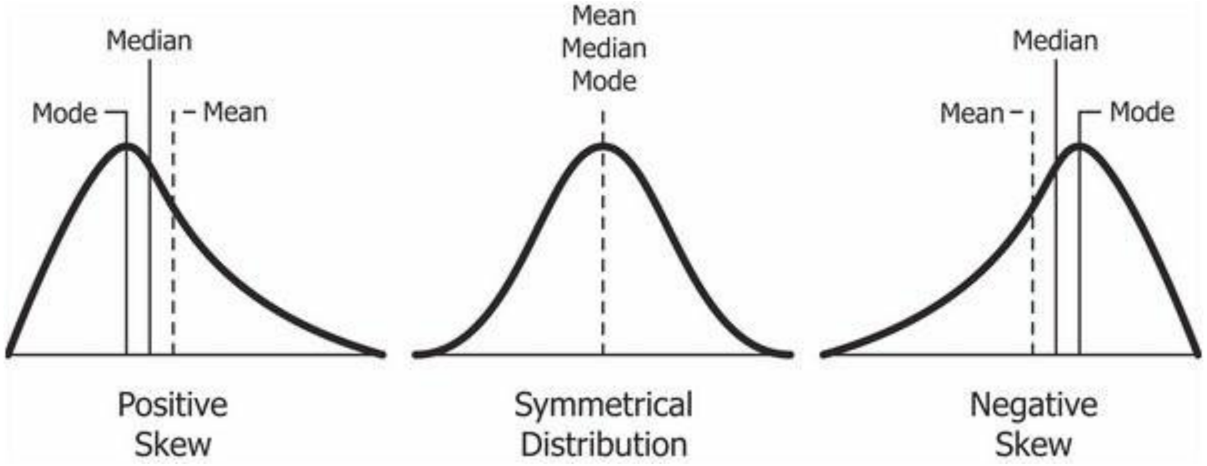
Skewness




```
df.skew()
```

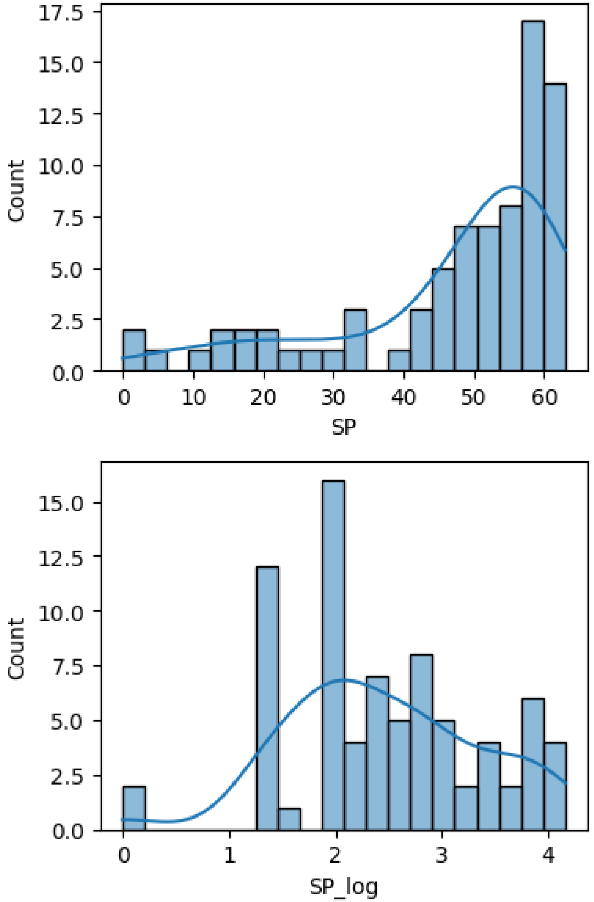
```
EA        -0.61
EB        -0.40
EC        -0.36
SP        -1.46
```

Correcting skewness




```
# correct the negative skewness of NEG_SKEW using log transformation
df['NEG_SKEW_log'] = np.log1p(df['NEG_SKEW'].max() - df['NEG_SKEW’])

# correct the positive skewness of POS_SKEW using log transformation
df['POS_SKEW_log'] = np.log1p(df['POS_SKEW'])
```

Correcting


17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Count
0
10
20
30
40
50
60
SP
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Count
0
1
2
3
4
SP_log

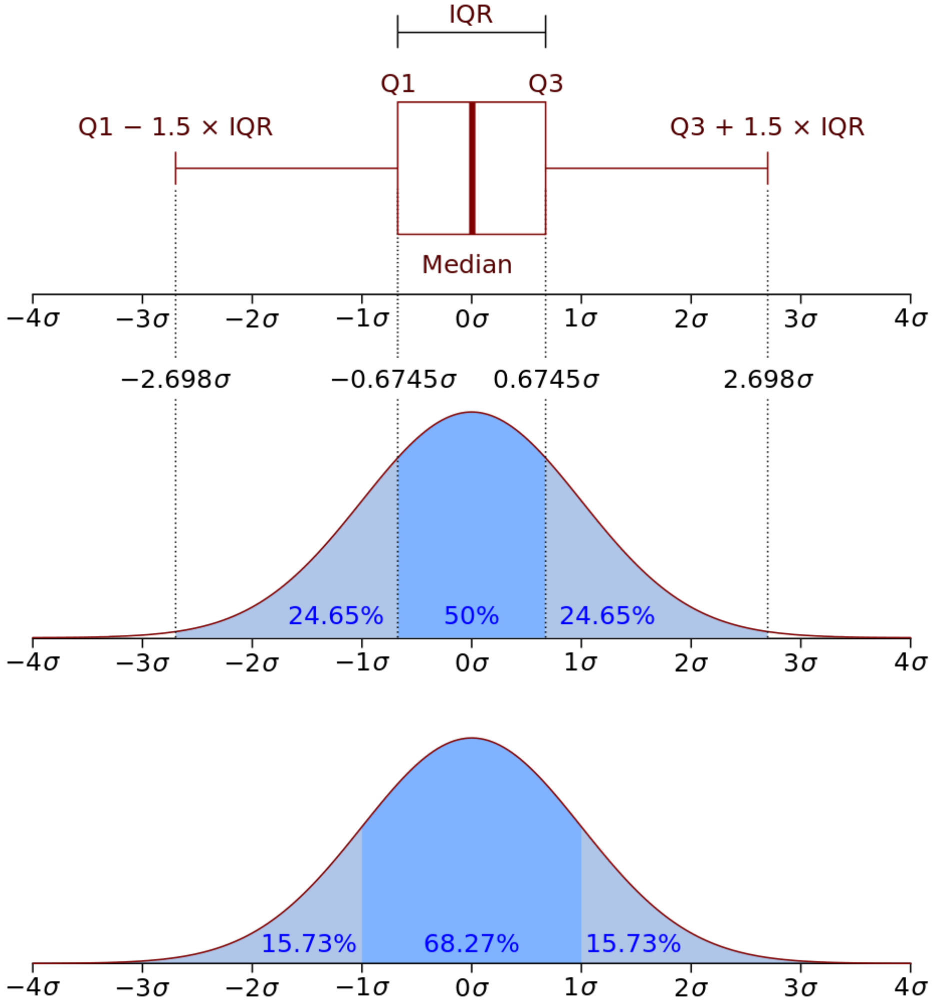
It's better to use scikit-learn for this!

```
pipeline = Pipeline(steps=[
("imputer", ColumnTransformer(...)),

("scaler", MinMaxScaler()), # scale the data to 0..1, so we don't have problems with log's

("log_transformer", ColumnTransformer( transformers=[

   ("COL_NEG", FunctionTransformer(func=lambda x: np.log1p(1.0 - x)), ["COL_NEG"]),

   ("COL_POS", FunctionTransformer(func=lambda x: np.log1p(x)), ["COL_POS"]),
  ],
  remainder="passthrough"
 )),
("regressor", LinearRegression()),
])
```

SAXION UNIVERSITY OF APPLIED SCIENCES

Boxplots

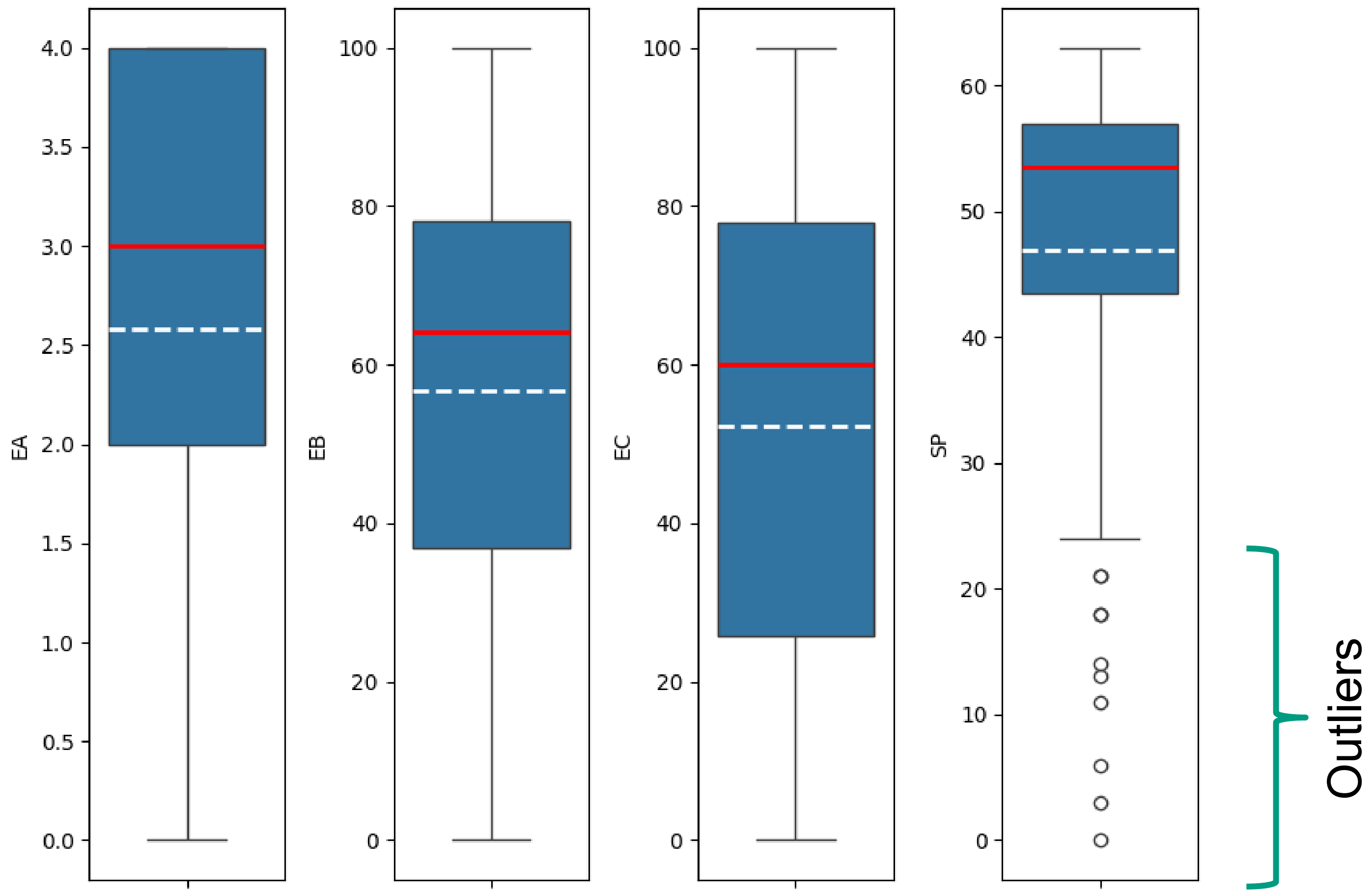
Interquartile range



Boxplots

```
fig,axs = plt.subplots(ncols=4, nrows=1, figsize=(8, 6))
index = 0
axs = axs.flatten()
for k, v in df[['EA', 'EB', 'EC', 'SP']].items():

   sns.boxplot(y=k, data=df, ax=axs[index],
                  medianprops={"color": "red", "linewidth": 2},
                  meanprops={"color": "white", "linewidth": 2},
                  meanline=True, showmeans=True)
   index += 1
plt.tight_layout()
```

Boxplots: outliers

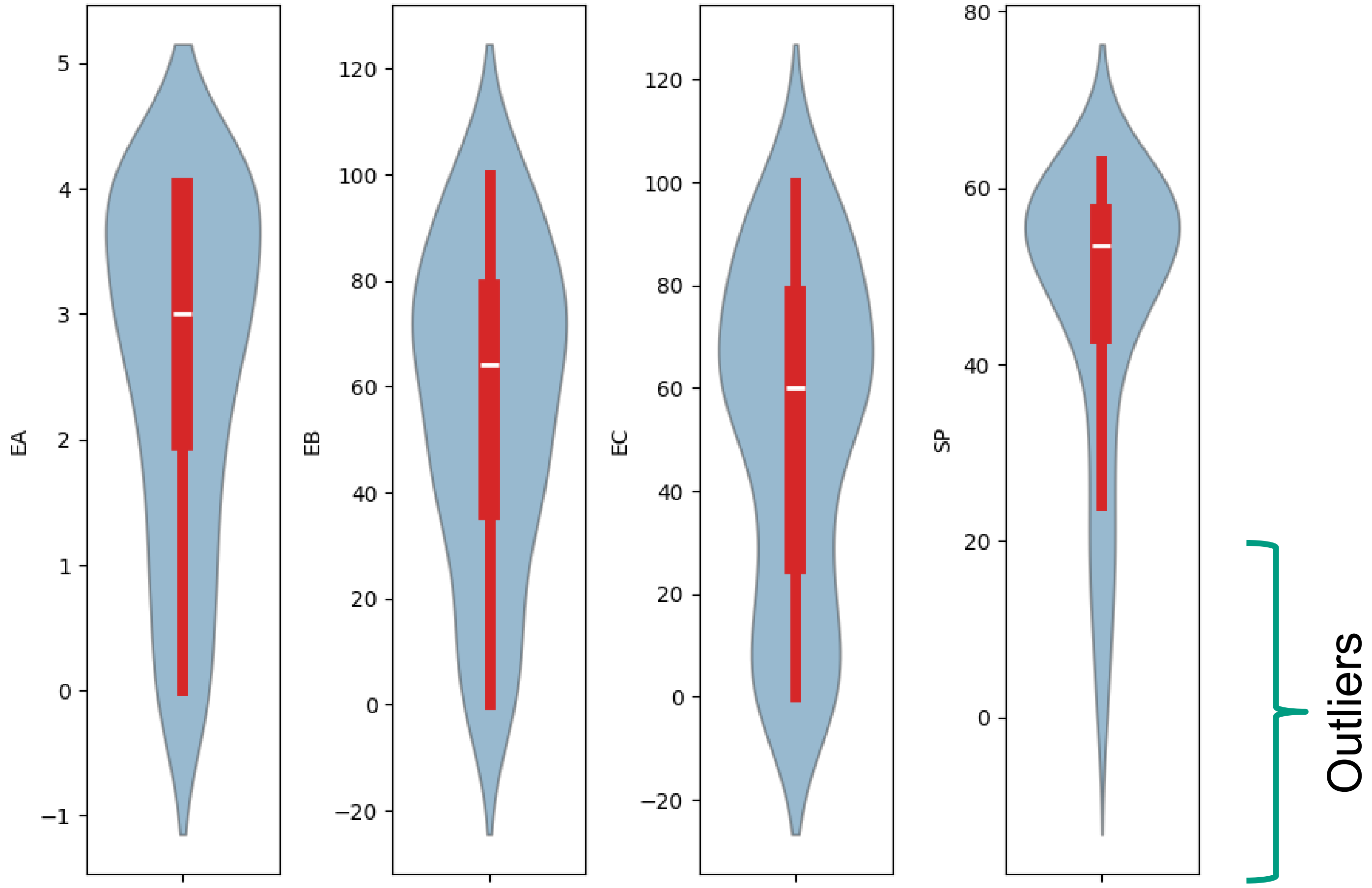


SAXION UNIVERSITY OF APPLIED SCIENCES

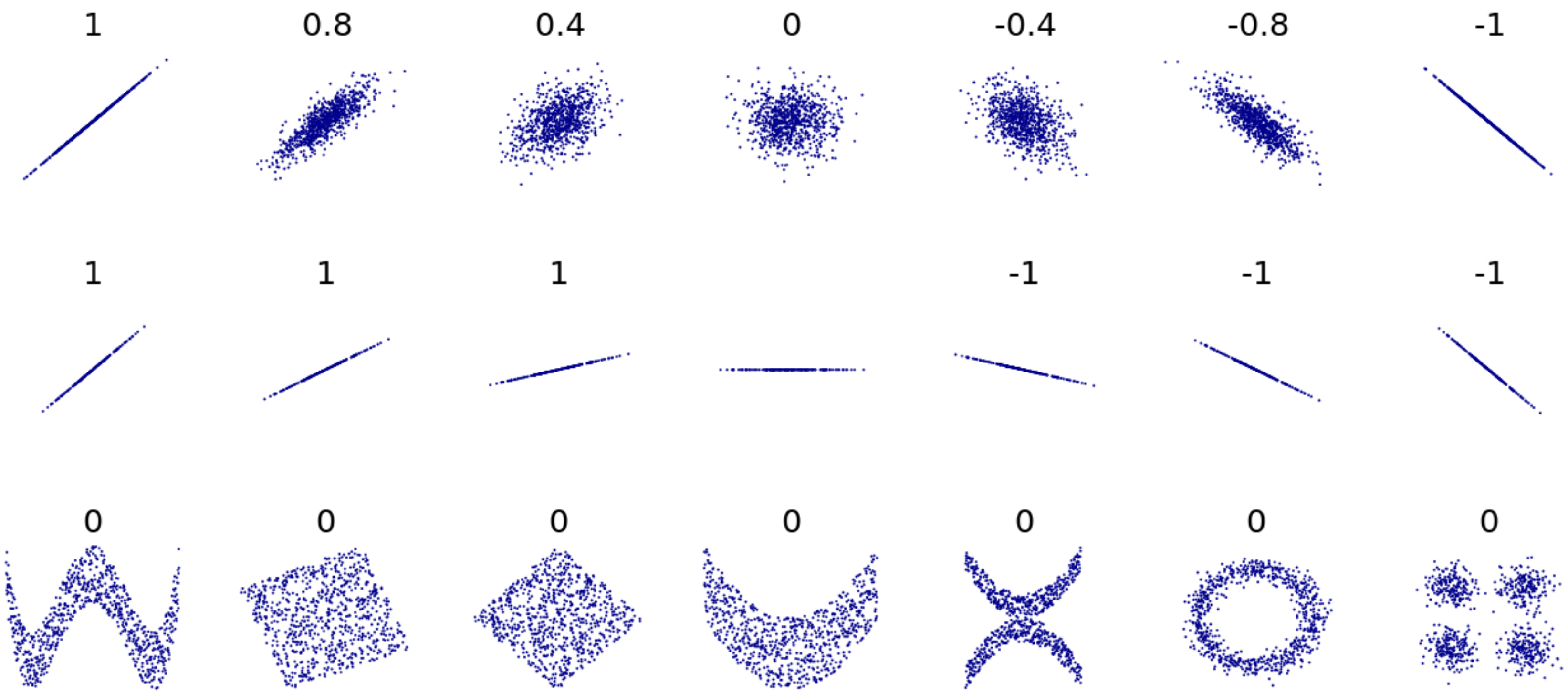
Violin plots

```
fig,axs = plt.subplots(ncols=4, nrows=1, figsize=(8, 6))
index = 0
axs = axs.flatten()
for k, v in df[['EA', 'EB', 'EC', 'SP']].items():

  sns.violinplot(y=k, data=df, ax=axs[index], alpha=0.5,
                 inner_kws={
                            "box_width": 10,
                            "whis_width": 5,
                            "color": sns.color_palette()[3]})
  index += 1
plt.tight_layout()
```

Violin plots: outliers

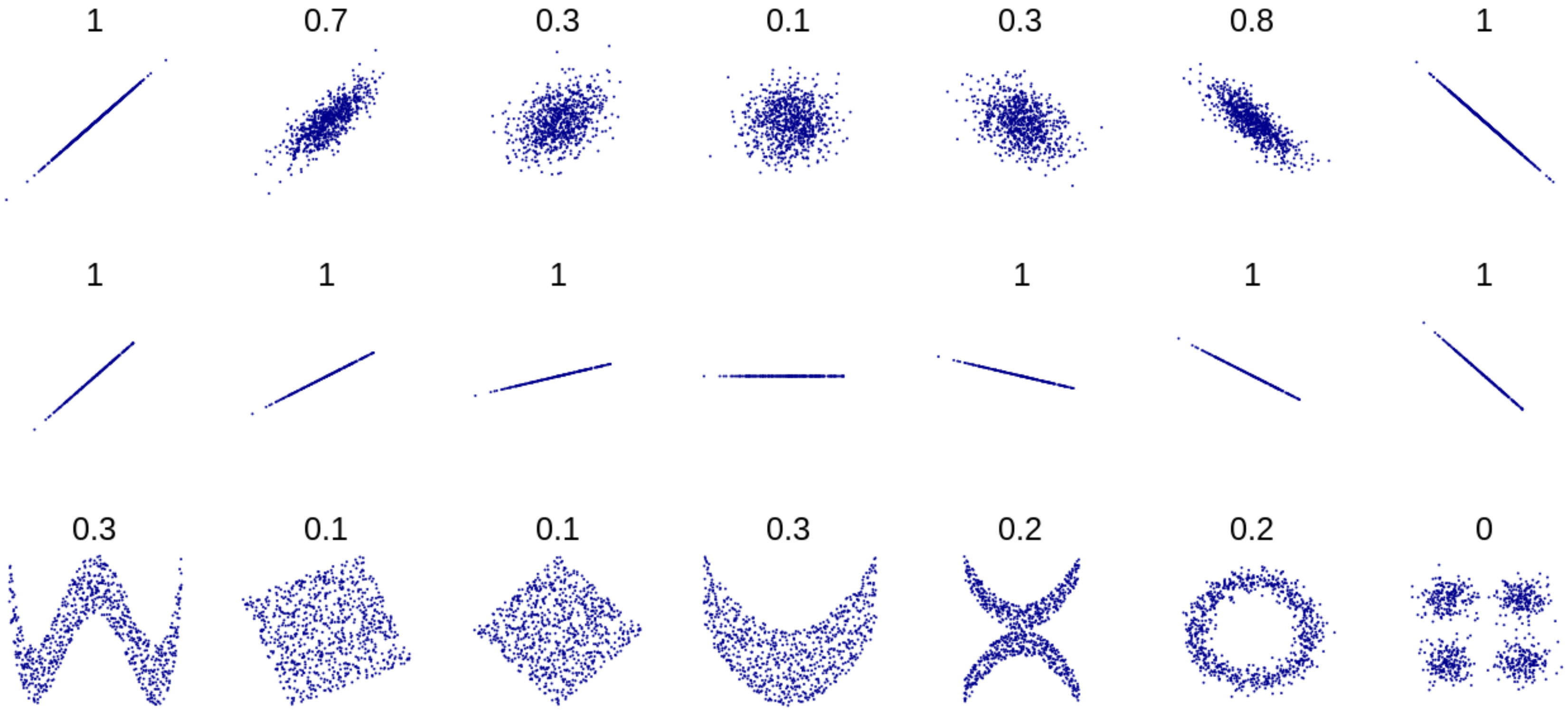


Pearson correlation coefficient



Works only for linear (y=ax+b) relationships!

Other correlations: distance correlation



To get it: `pip install dcor`

https://dcor.readthedocs.io/en/stable/index.html

SAXION UNIVERSITY OF APPLIED SCIENCES

Looking at correlations

```
pd.set_option('display.precision', 2)

pd.DataFrame({
    'Pearson': df.corr(method='pearson')['SP'],
    'Spearman': df.corr(method='spearman')['SP']
})
```

Correlation:

- ~0: none at all
- <0.5: none ÷ weak
- >0.5: it seems data is correlated
- >0.8: strong

Looking at correlations

	Pearson	Spearman
EA	0.55	0.64
EB	0.72	0.71
EC	0.80	0.81
SP	1.00	1.00
HSP_B	0.62	0.75
HSP_E	-0.38	-0.28
HSP_G	0.07	-0.17
HSP_K	-0.52	-0.51
HSP_N	-0.35	-0.27

Correlation:

~0: none at all
<0.5: none ÷ weak
>0.5: it seems data is correlated
>0.8: strong

SAXION UNIVERSITY OF APPLIED SCIENCES

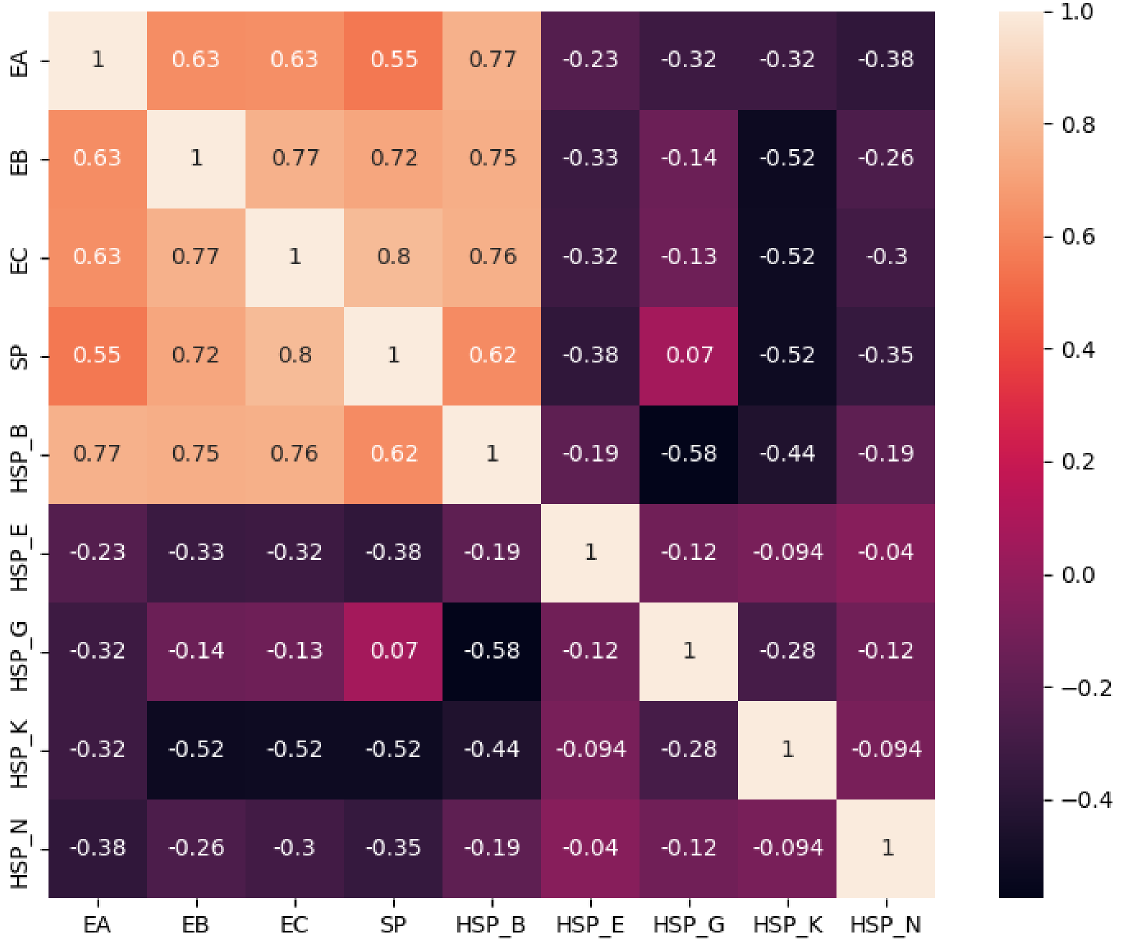
Looking at correlations

```python
import seaborn as sns

corr = df.corr()
plt.figure(figsize = (8, 6))
sns.heatmap(corr, square=True, annot=True)
plt.tight_layout()
```

Some features are colinear:

- EA~HSP_B
- EB~[EC, HSP_B]
- EC~HSP_B

Colinear features are candidates for removal

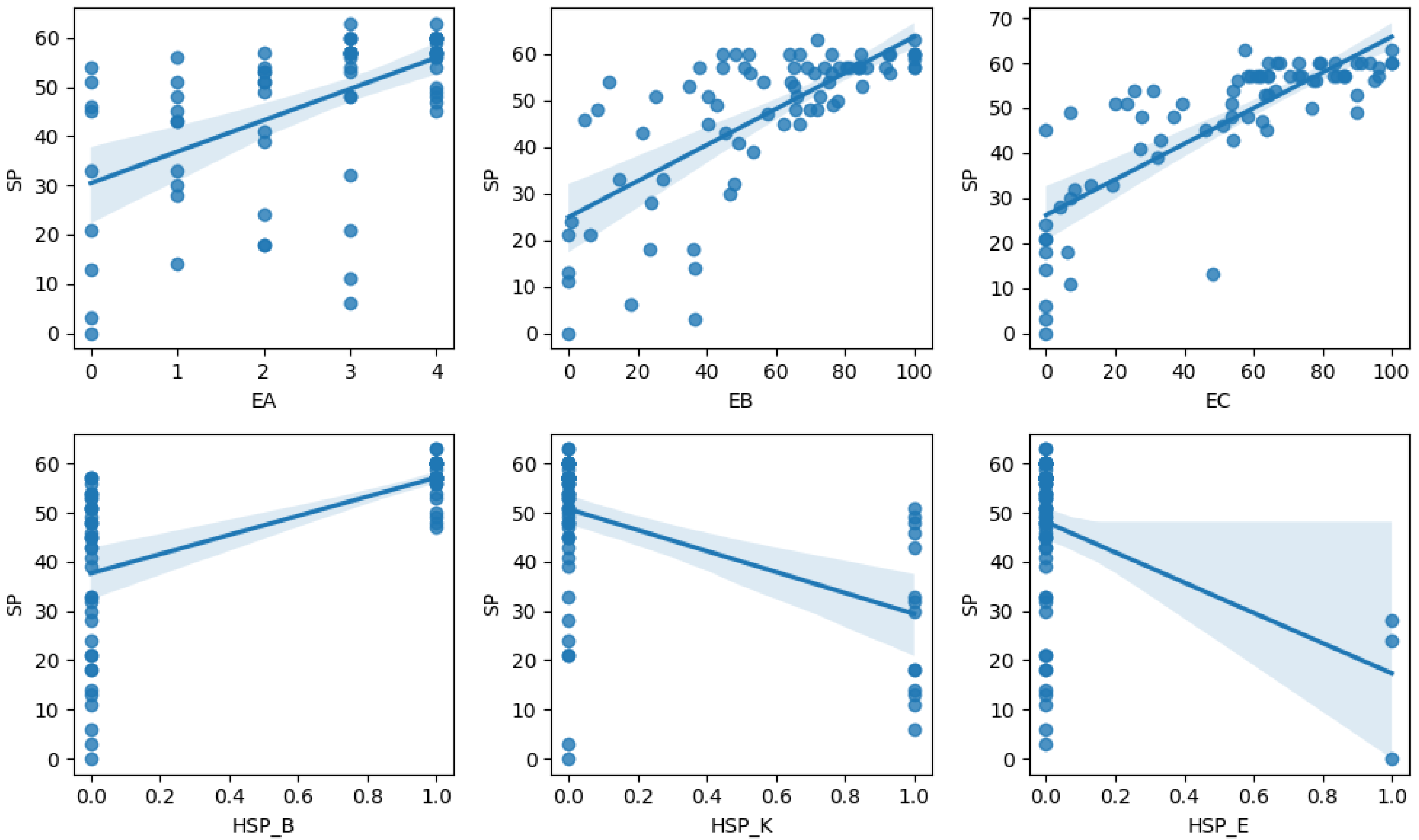


SAXION UNIVERSITY OF APPLIED SCIENCES

Regression plots

```
columns = ['EA', 'EB', 'EC', 'HSP_B', 'HSP_K', 'HSP_E']

fig, axs = plt.subplots(ncols=3, nrows=2, figsize=(10, 6))

index = 0
axs = axs.flatten()

for i, (k, v) in enumerate(df[columns].items()):

    sns.regplot(y=df['SP'], x=v, ax=axs[i])

plt.tight_layout()
```

Regression plots

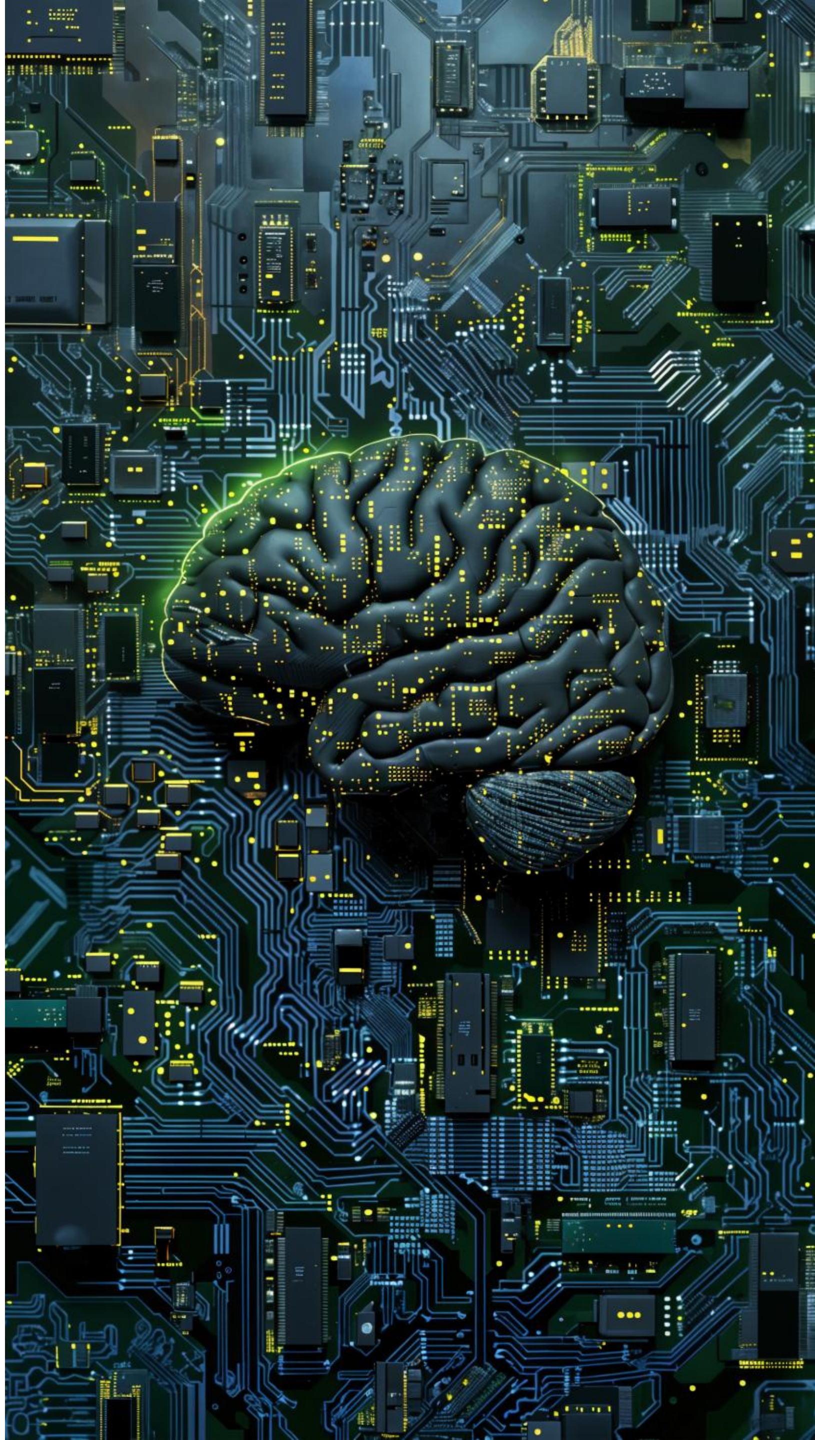


SAXION UNIVERSITY OF APPLIED SCIENCES

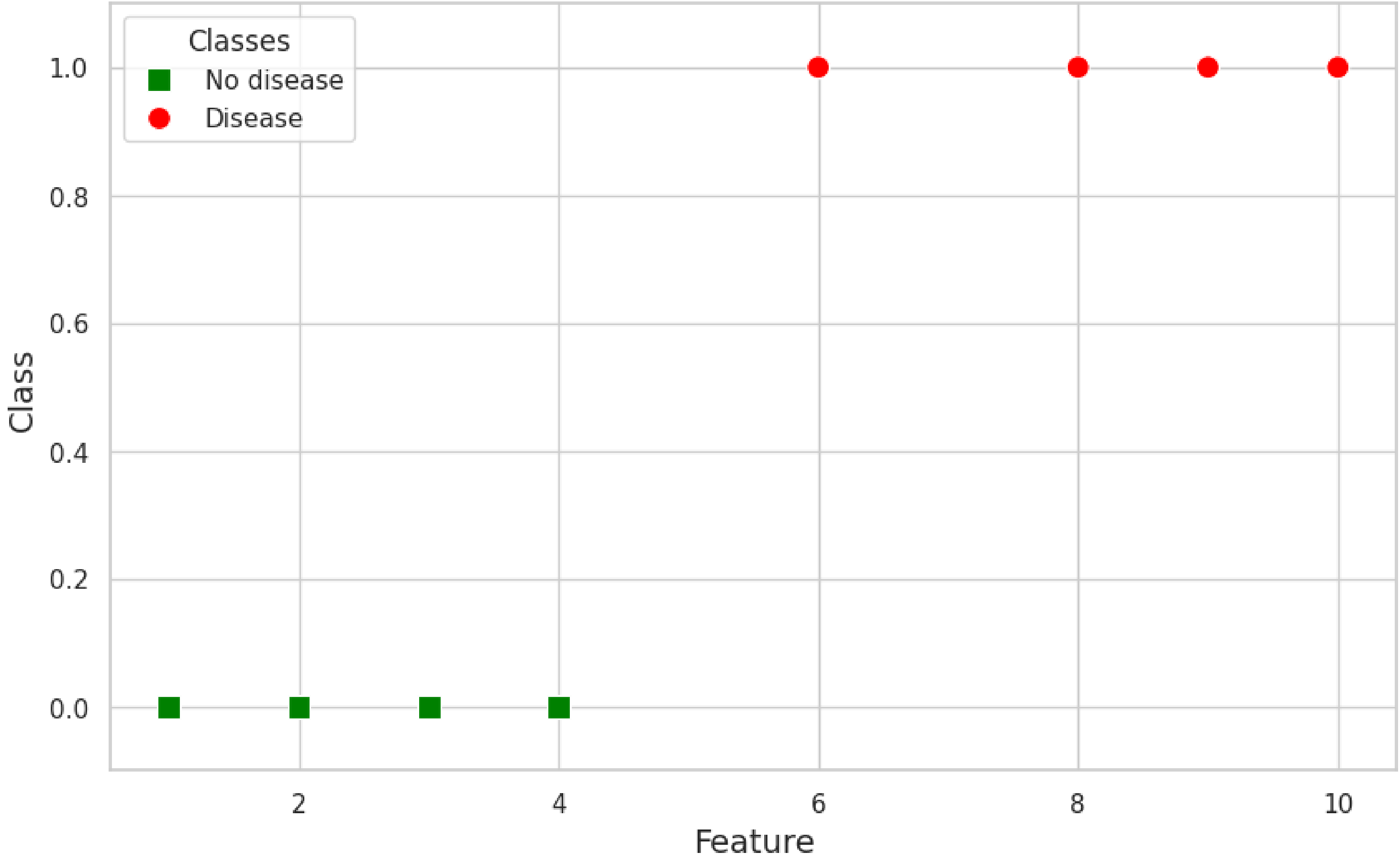
Work, work, work

Classification

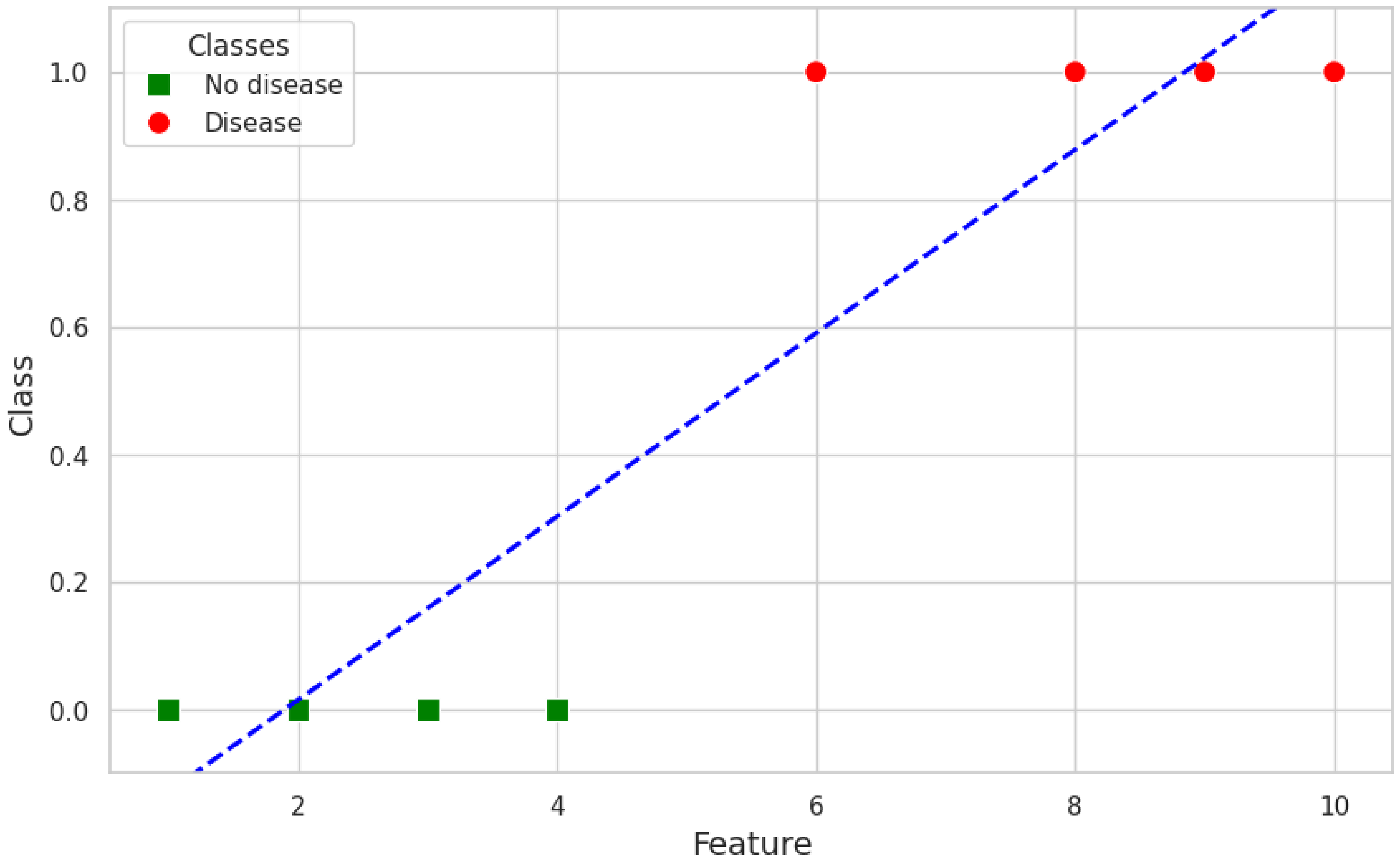
Logistic regression



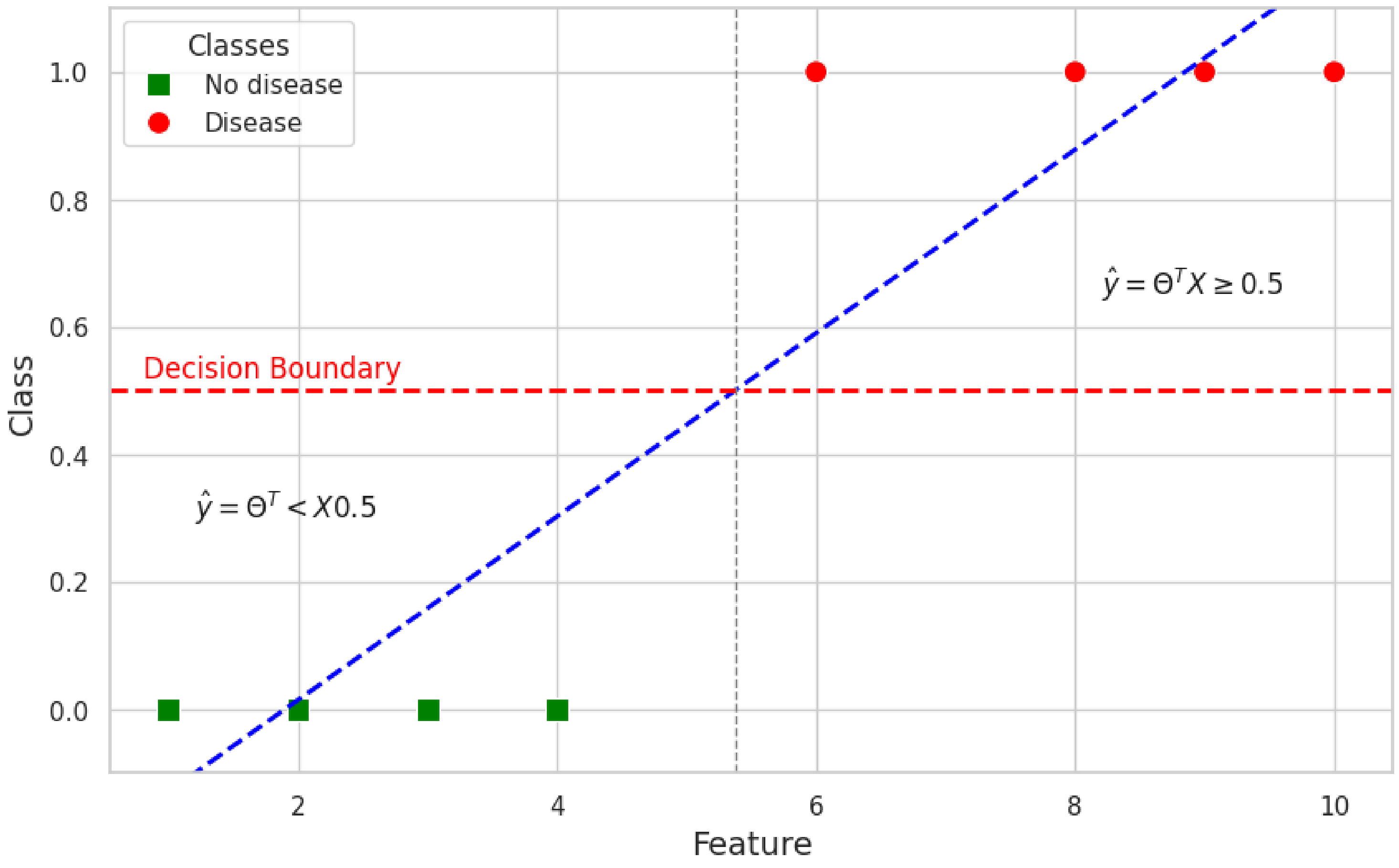
Simple regression problem



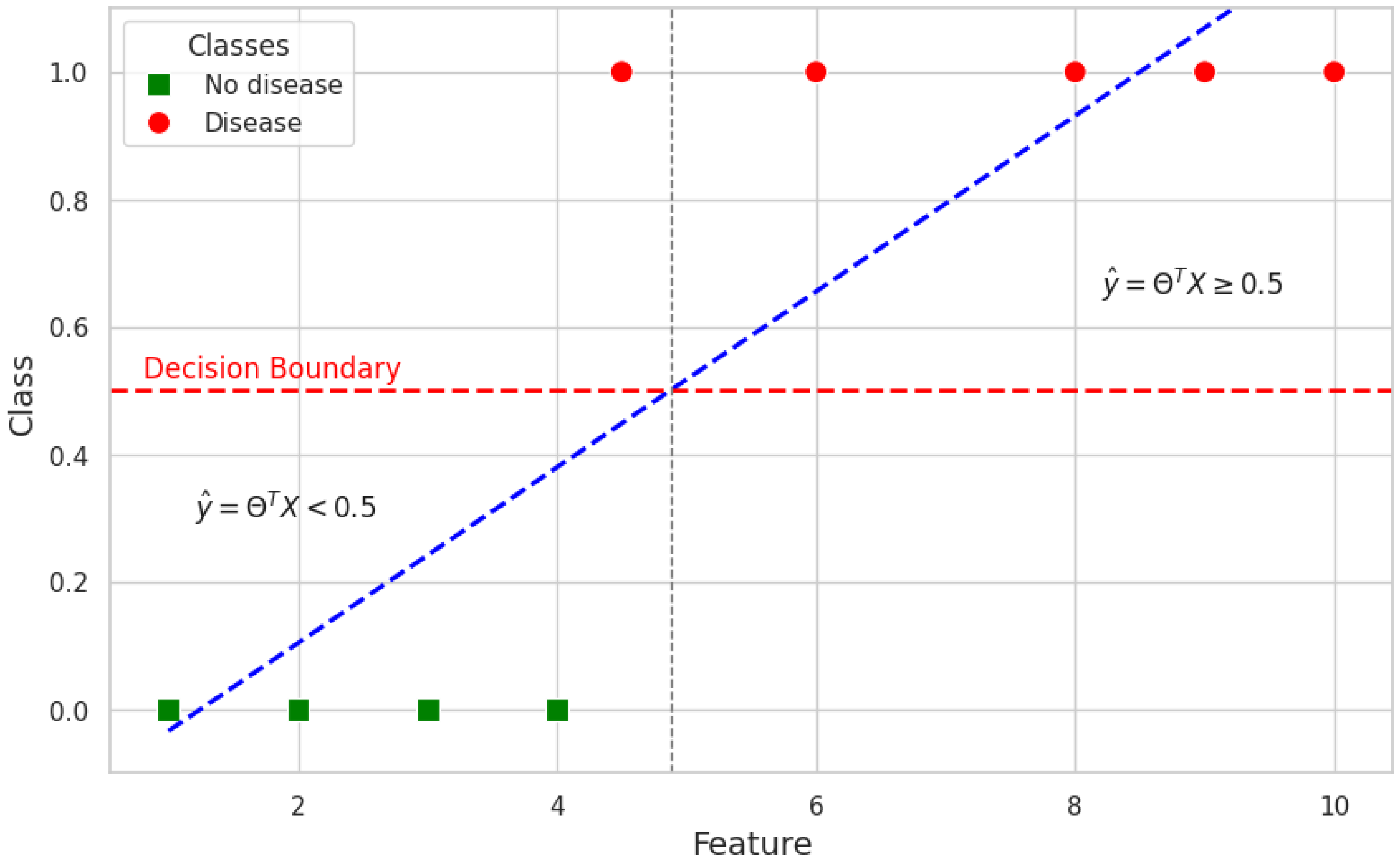
Simple regression problem



Simple regression problem

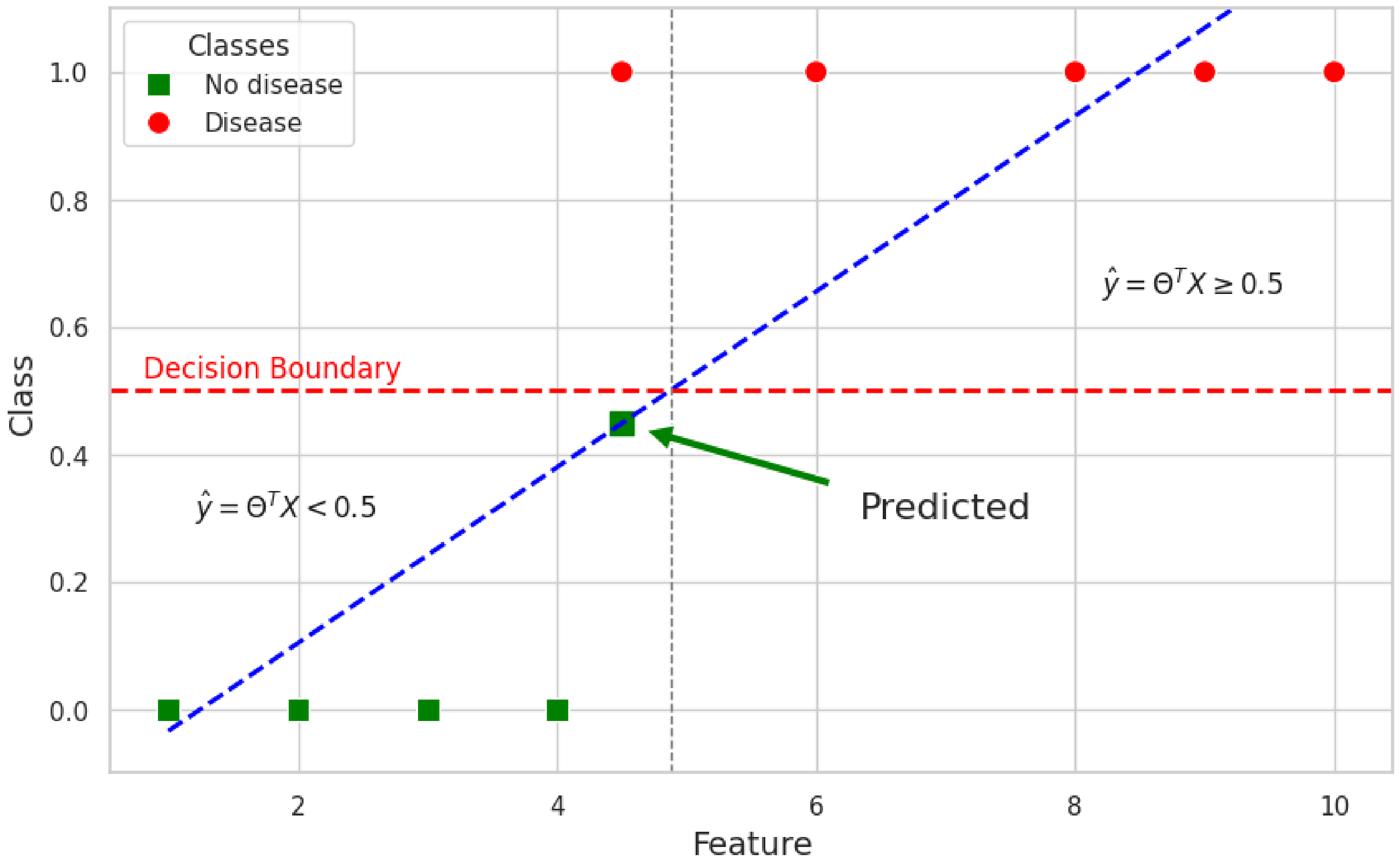


Simple regression problem

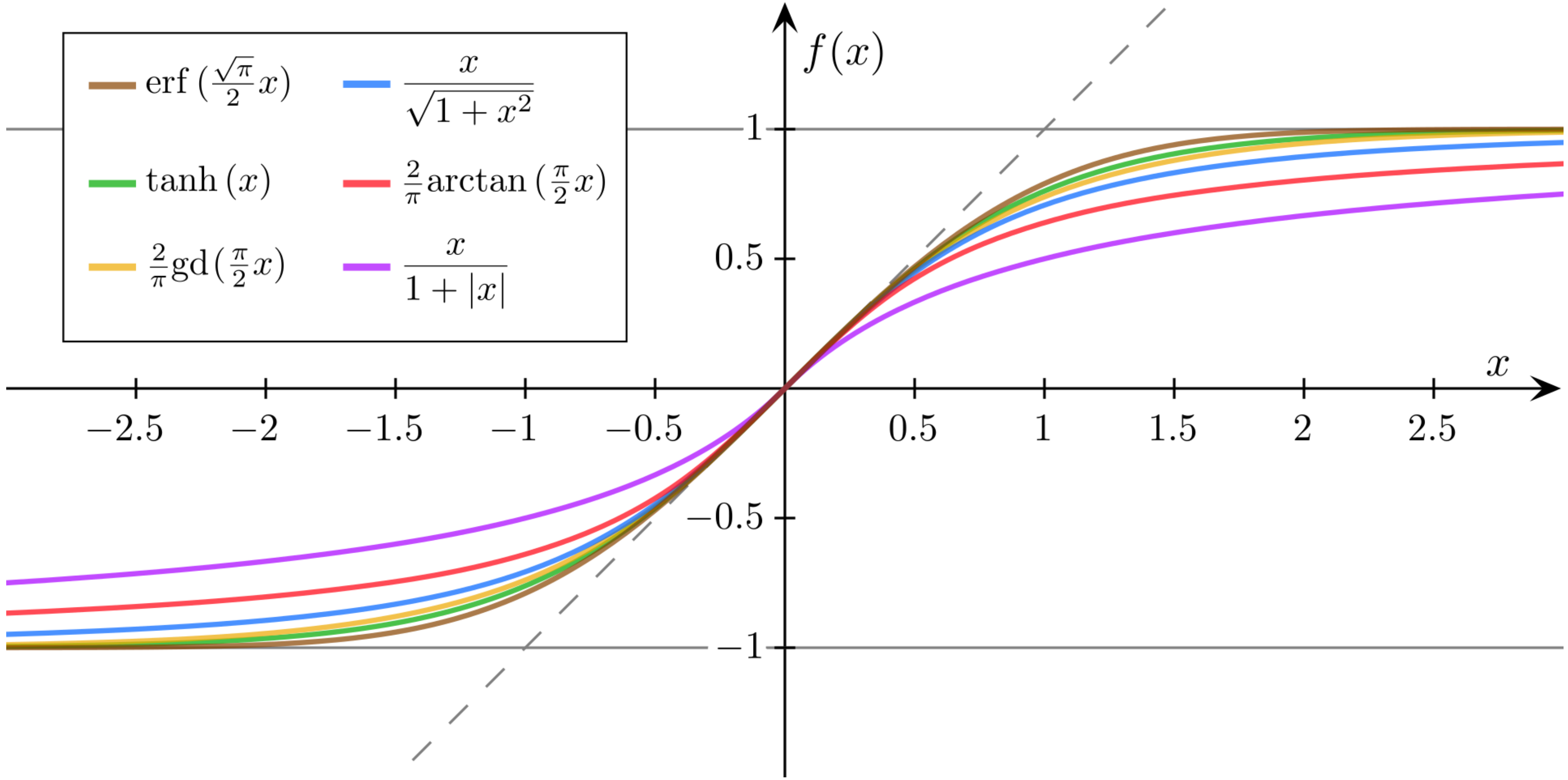


SAXION UNIVERSITY OF APPLIED SCIENCES

Simple regression problem



Welcome the sigmoid

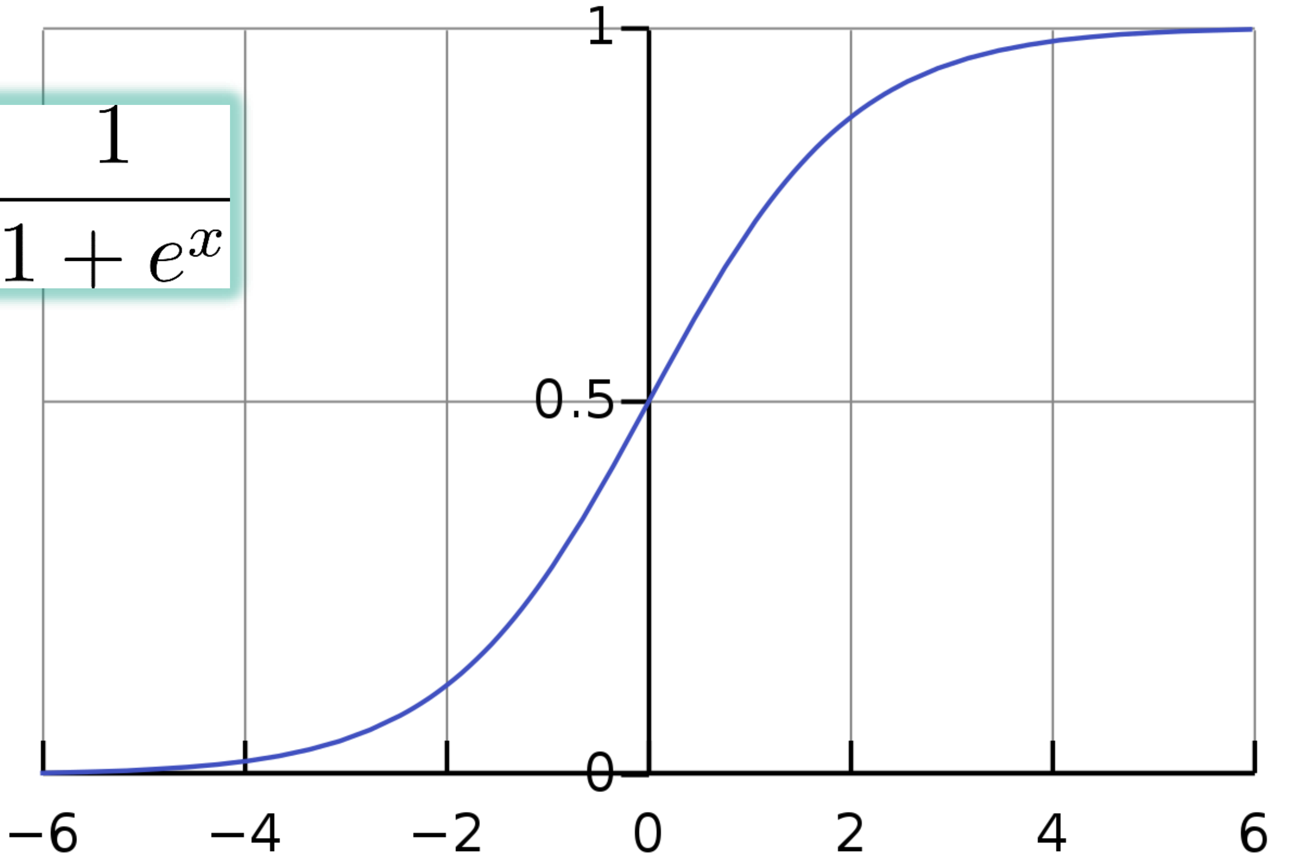


By Georg-Johann (adapted from Geek3), CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=11498624

SAXION UNIVERSITY OF APPLIED SCIENCES

Logistic function

$$f(x) = \frac{1}{1 + e^{x}}$$



By Qef - Created from scratch with gnuplot, Public Domain, https://commons.wikimedia.org/w/index.php?curid=4310325

Logistic regression

- Linear Regression:

$$h_\Theta(X) = \Theta^T X$$

- Logistic Function:

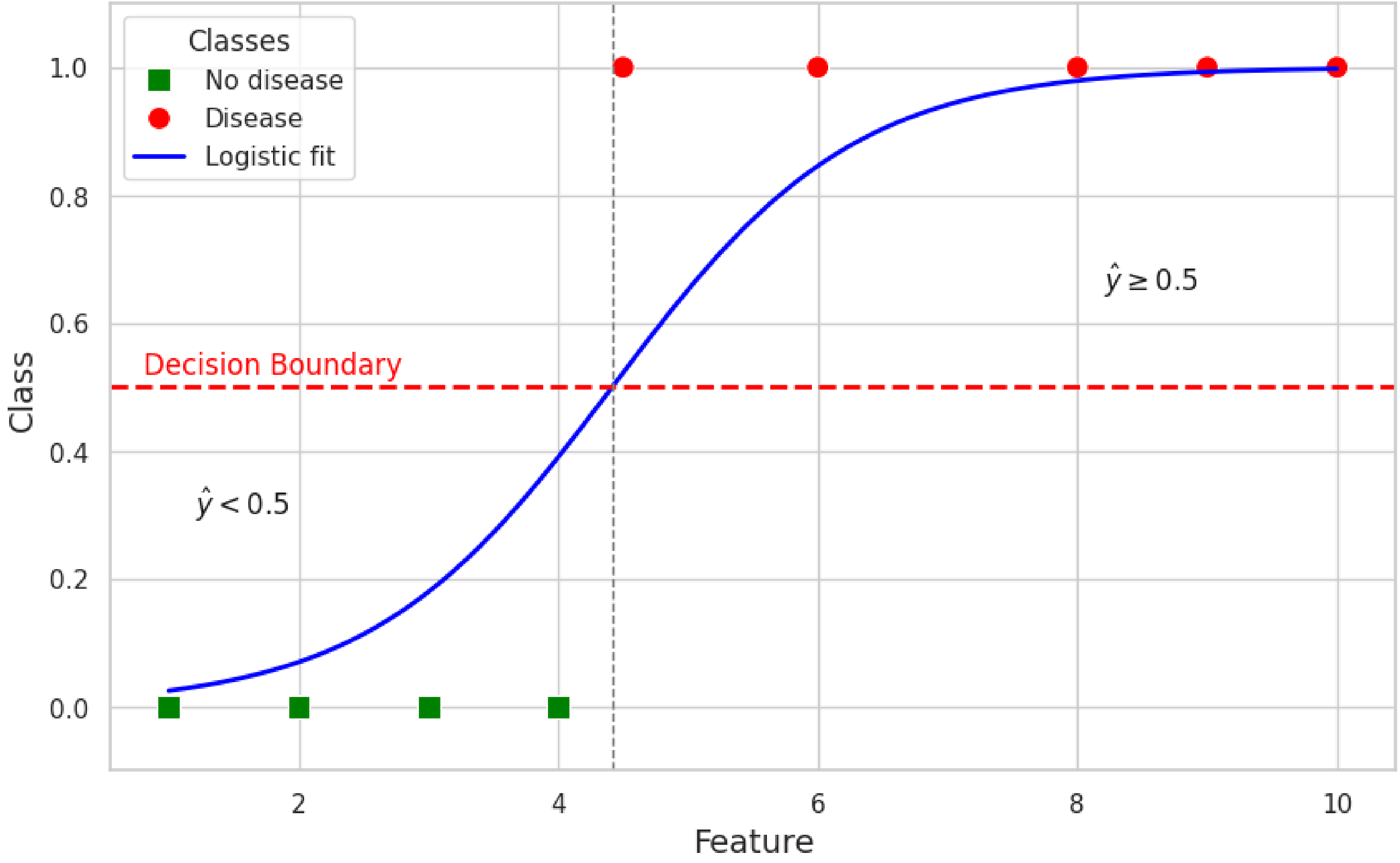
$$f(x) = \frac{1}{1 + e^x}$$

- Logistic Regression:

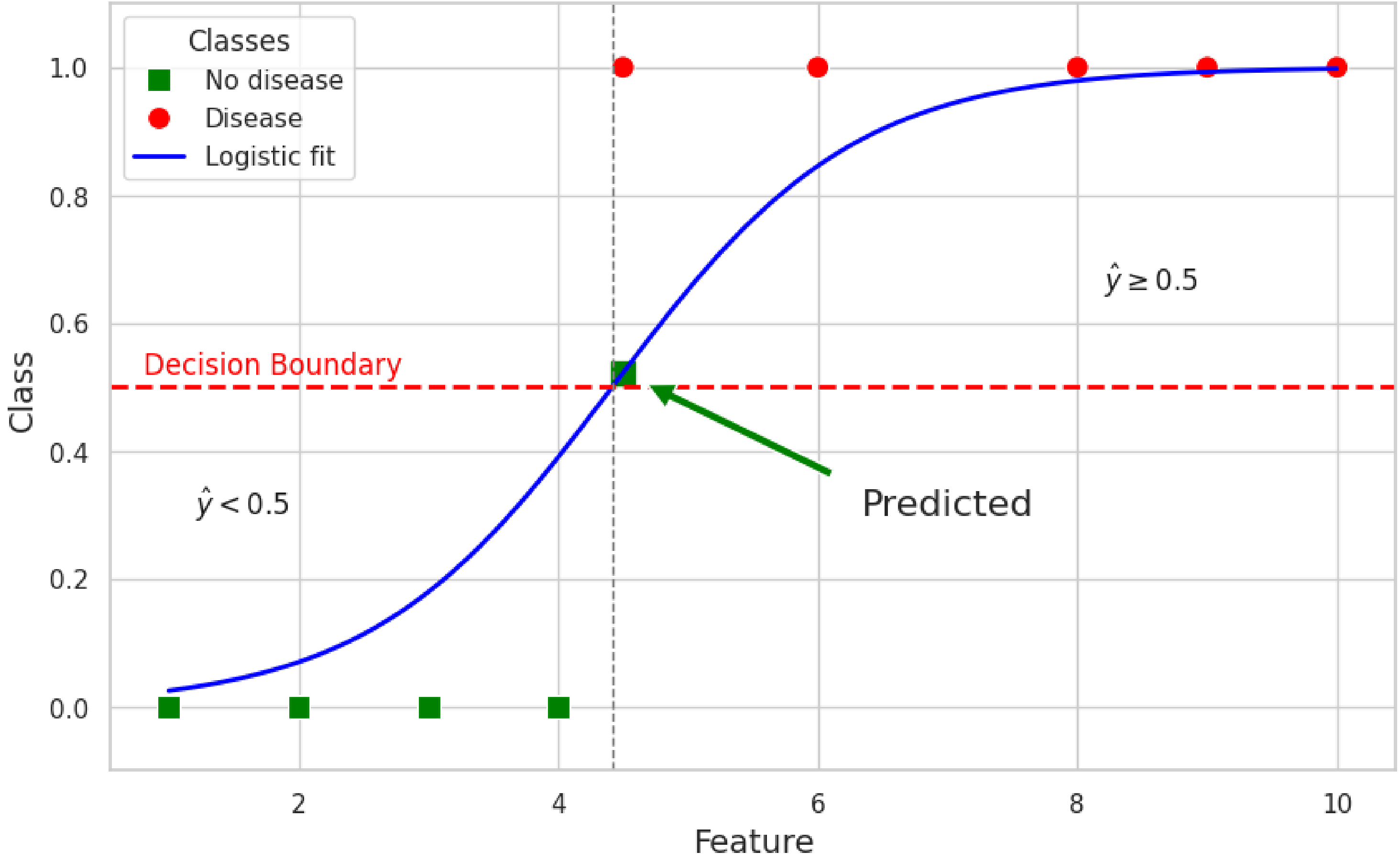
$$h_\Theta(X) = \frac{1}{1 + e^{\Theta^T X}}$$

SAXION UNIVERSITY OF APPLIED SCIENCES

Simple regression problem



Simple regression problem



SAXION UNIVERSITY OF APPLIED SCIENCES

Logistic Regression

University of California, Irvine, *Heart Disease Data Set*

```
data = read_csv("heart.csv")
data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Logistic Regression

University of California, Irvine, *Heart Disease Data Set*

```
data.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	303.00	303.00	303.00	303.00	303.00	303.00	303.00	303.00	303.00	303.00	303.00	303.00	303.00	303.00
mean	54.37	0.68	0.97	131.62	246.26	0.15	0.53	149.65	0.33	1.04	1.40	0.73	2.31	0.54
std	9.08	0.47	1.03	17.54	51.83	0.36	0.53	22.91	0.47	1.16	0.62	1.02	0.61	0.50
min	29.00	0.00	0.00	94.00	126.00	0.00	0.00	71.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	47.50	0.00	0.00	120.00	211.00	0.00	0.00	133.50	0.00	0.00	1.00	0.00	2.00	0.00
50%	55.00	1.00	1.00	130.00	240.00	0.00	1.00	153.00	0.00	0.80	1.00	0.00	2.00	1.00
75%	61.00	1.00	2.00	140.00	274.50	0.00	1.00	166.00	1.00	1.60	2.00	1.00	3.00	1.00
max	77.00	1.00	3.00	200.00	564.00	1.00	2.00	202.00	1.00	6.20	2.00	4.00	3.00	1.00

Logistic Regression

University of California, Irvine, *Heart Disease Data Set*

```
data.corr(method="pearson")['target']
```

```
age        -0.23
sex        -0.28
cp          0.43
thalach     0.42
exang      -0.44
oldpeak    -0.43
slope       0.35
ca         -0.39
thal       -0.34
target      1.00
Name: target, dtype: float64
```

Logistic Regression

```python
from sklearn.linear_model import LogisticRegression


X = data[['thalach']]
y = data['target']

log_reg = LogisticRegression(solver="lbfgs", max_iter=100)

log_reg.fit(X, y)
```

Class probabilities

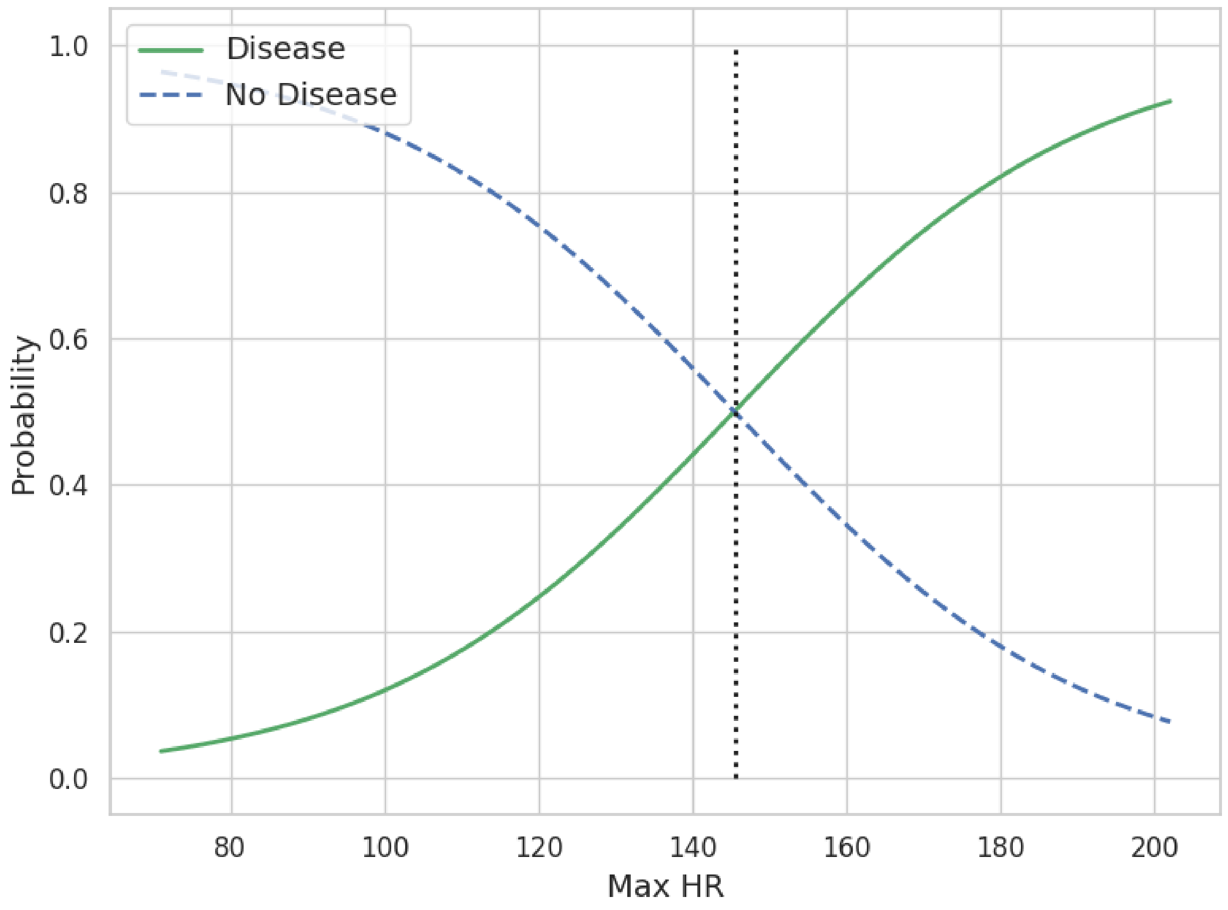
```
from sklearn.model_selection import cross_val_score

cross_val_score(log_reg, X, y, cv=5, scoring="accuracy").mean()
>> 0.7030054644808743
```

```
log_reg.predict_proba(X)
```

$$h_\Theta(X) = \frac{1}{1 + e^{\theta_1 x_1 + \theta_0}}$$

X	0	1
0	0.45	0.55
50	0.46	0.54
…		
100	0.19	0.81
…		
150	0.58	0.42
…		

One variable regression

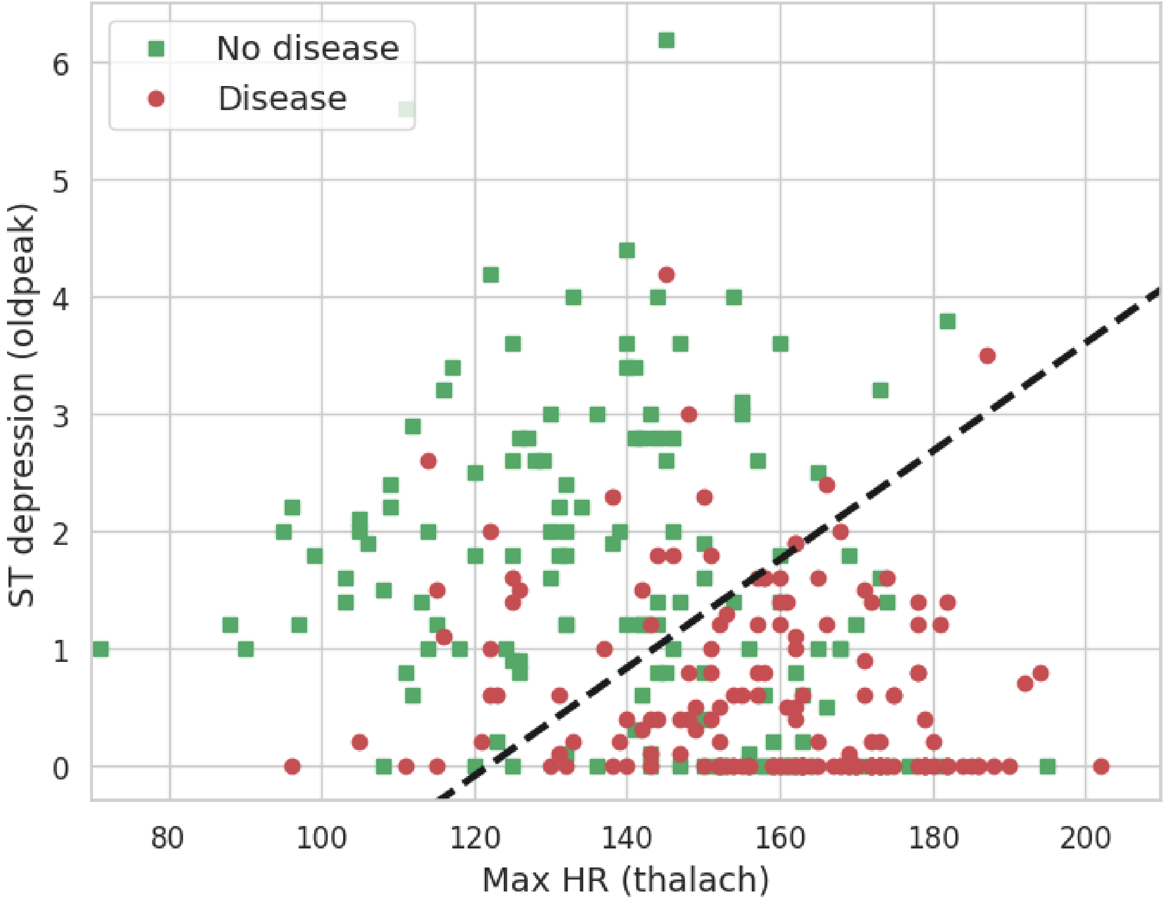


Two variables regression

```
X = data[['thalach', 'oldpeak']]
y = data['target']


log_reg = LogisticRegression(solver="lbfgs", C=10, max_iter=1000)
log_reg.fit(X, y)

>> 0.7325683060109289
```

Plot predicted classes vs. the variables
Plot decision boundary

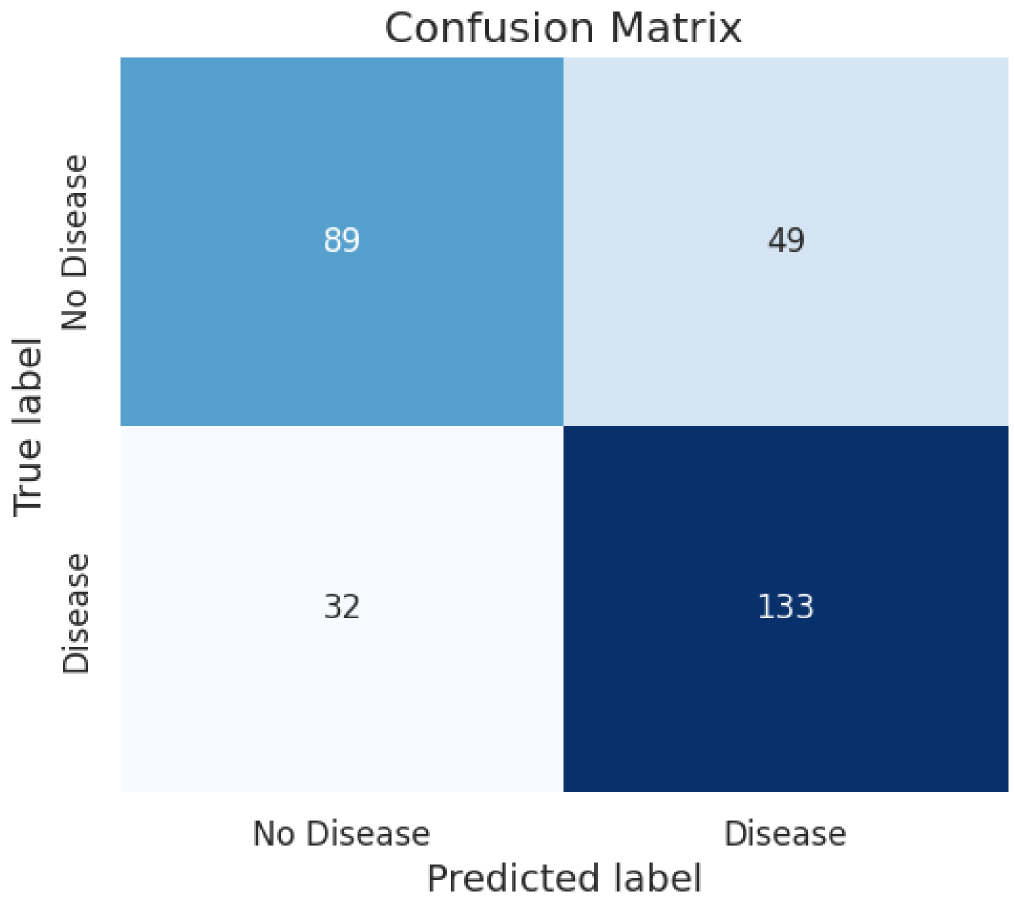
Two variables regression



Confusion Matrix

```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix

y_pred = cross_val_predict(log_reg, X, y, cv=5)

confusion_matrix(y, y_pred)


>> array([[ 89,  49],
          [ 32, 133]])
```

Confusion Matrix



Confusion Matrix

PREDICTED LABELS

TRUE LABELS	Negative	Positive
Negative	True Negative	False Positive
Positive	False Negative	True Positive

recall

precision

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Confusion Matrix

		PREDICTED	
		Negative	Positive
TRUE	Negative	89	49
	Positive	32	133

```
from sklearn.metrics import precision_score, recall_score

precision_score(y, y_pred)
recall_score(y, y_pred)

>> 0.73
>> 0.81
```

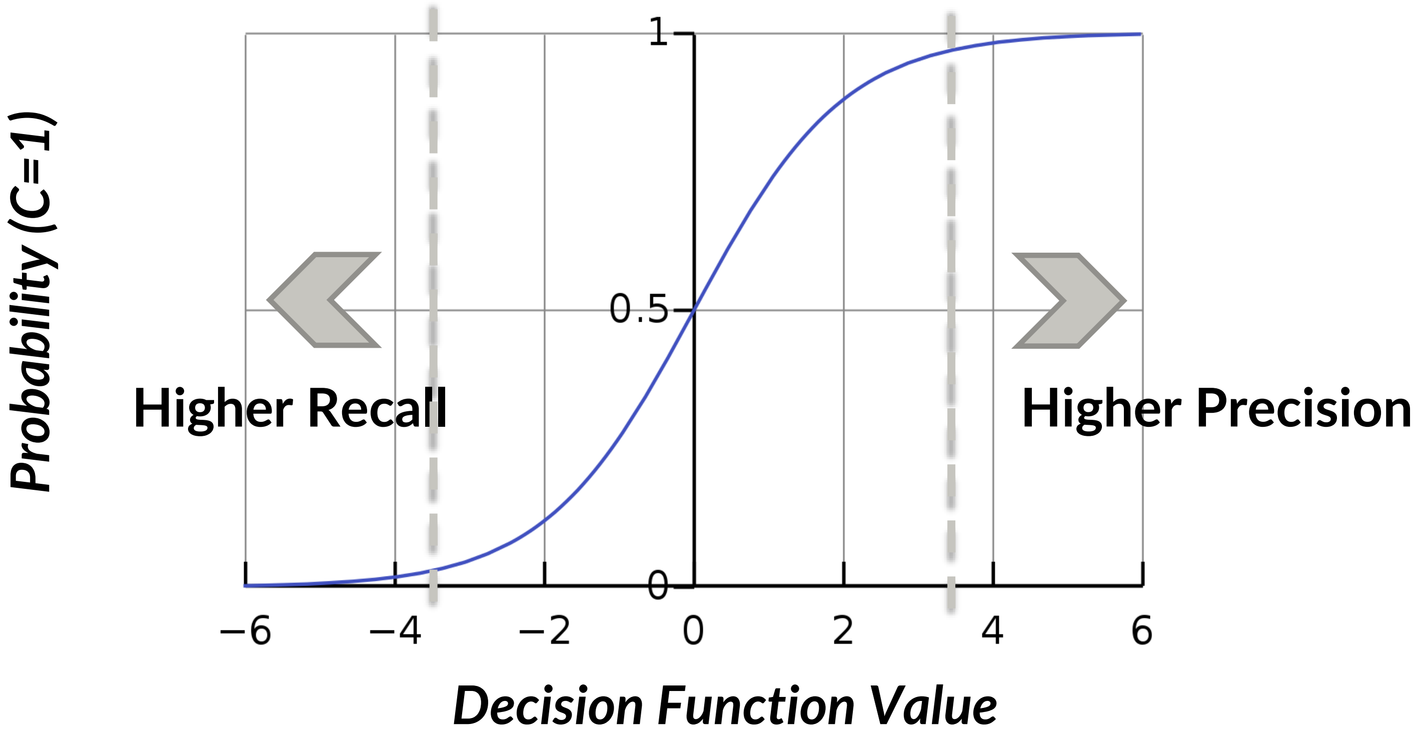
SAXION UNIVERSITY OF APPLIED SCIENCES

Decision Function

Probability of Positive class:

$$h_\Theta(X) = \frac{1}{1 + e^{\boxed{\Theta^T X}}}$$

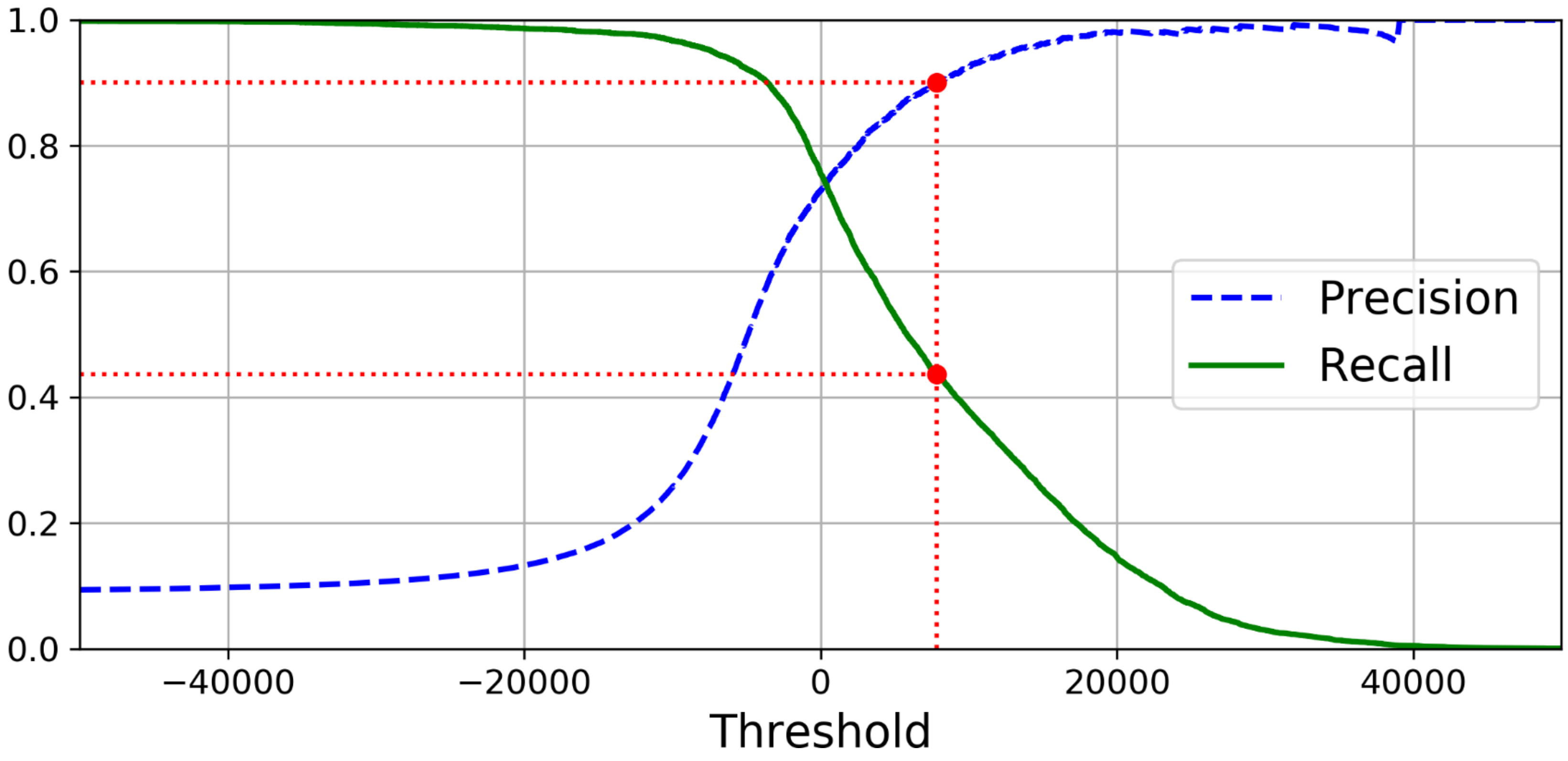
Decision Function



Decision Function Treshold

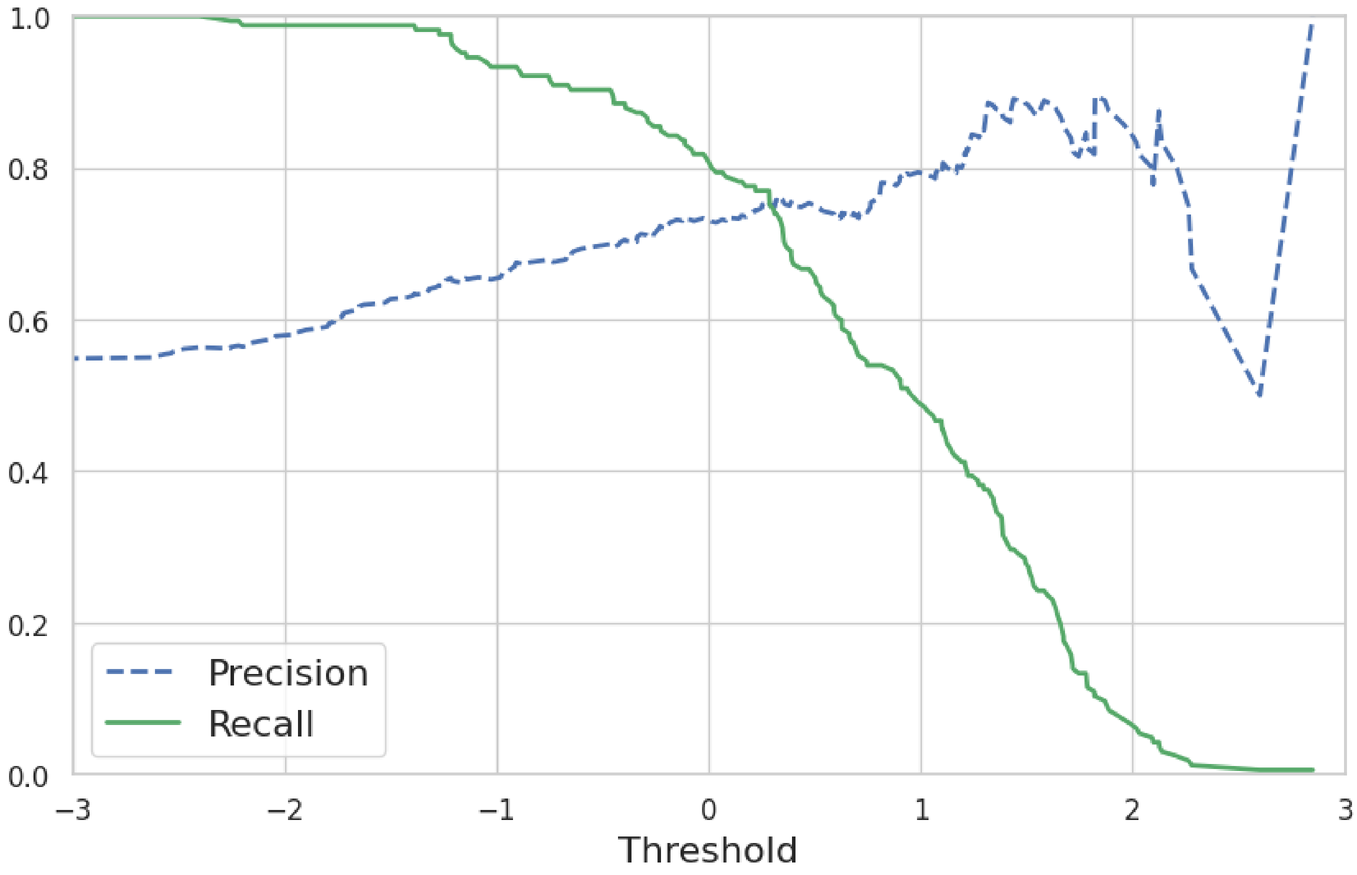
```python
y_scores = cross_val_predict(log_reg, X, y, cv=5, method="decision_function")


from sklearn.metrics import precision_recall_curve

precisions, recalls, thresholds = precision_recall_curve(y, y_scores)

# + some more code to print the precisions vs. thresholds
#   and recalls vs. thresholds
```

Precision vs. Recall



That's the ideal case…
For our heart disease dataset expect…

Precision vs. Recall



$$Precision = \frac{True\ Positives}{Classified\ as\ Positives}$$

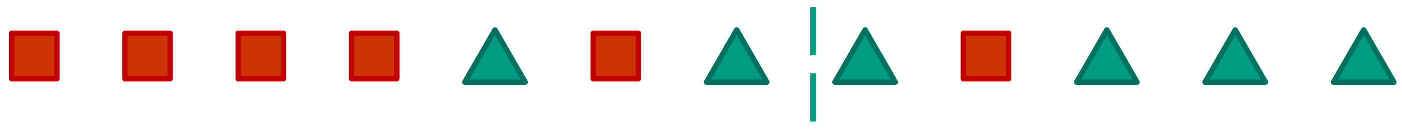
$$Recall = \frac{True\ Positives}{All\ Positives}$$

SAXION UNIVERSITY OF APPLIED SCIENCES

Precision vs. Recall

$$Precision = \frac{Positives\ Classified\ as\ Positives}{Classified\ as\ Positives}$$

Precision 0.5 0.54 0.6 0.67 0.75 0.7 0.83 0.8 0.75 1 1 1 1



Recall 1 1 0.83 0.83 0.67 0.5 0.5 0.33 0.17 0

$$Recall = \frac{Positives\ Classified\ as\ Positives}{All\ Positives}$$

But there is more! TPR and FPR

$$recall = TRP = \frac{TP}{TP + FN} = \frac{Positives\ Classified\ \text{as}\ Positives}{All\ Positives}$$

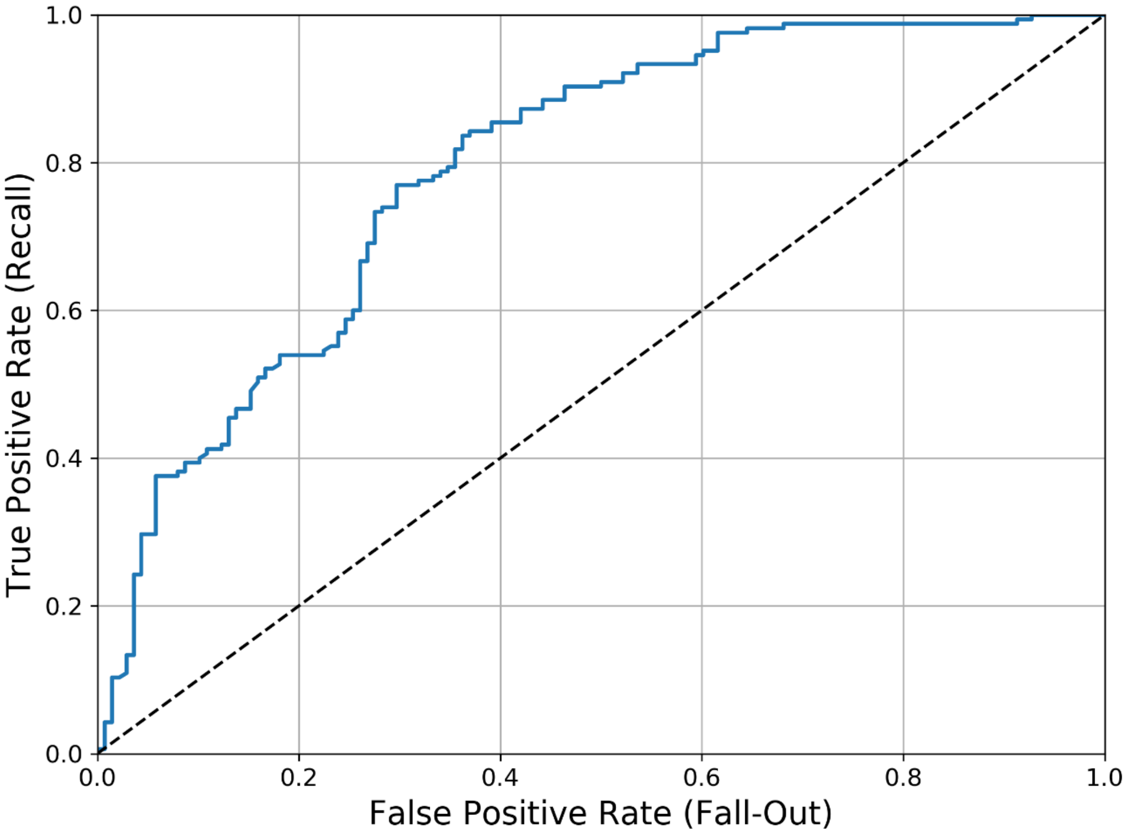
$$fallout = \ FPR = \frac{FP}{TN + FP} = \frac{Misclassified\ Nagatives}{All\ Negatives}$$

**True Positive Rate, False Positive Rate*

SAXION UNIVERSITY OF APPLIED SCIENCES

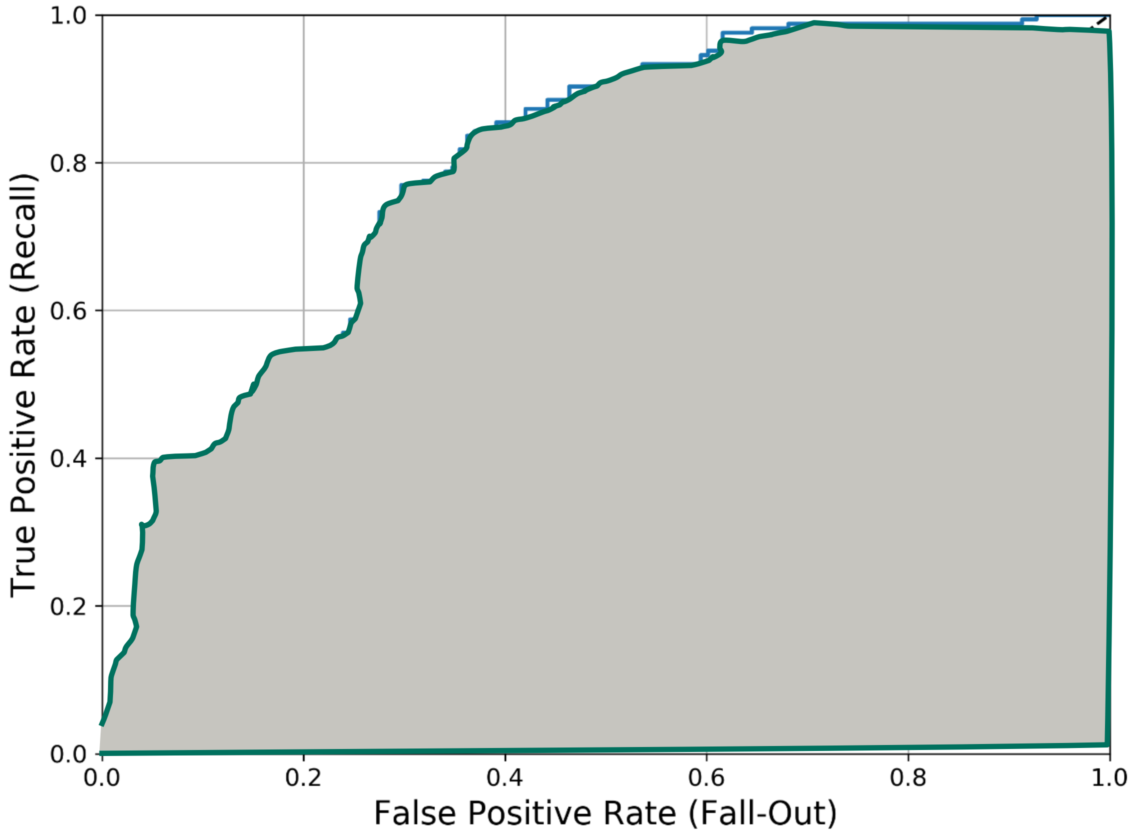
ROC curve

```
from sklearn.metrics import roc_curve

fpr, tpr, thres = roc_curve(y, y_scores)
```



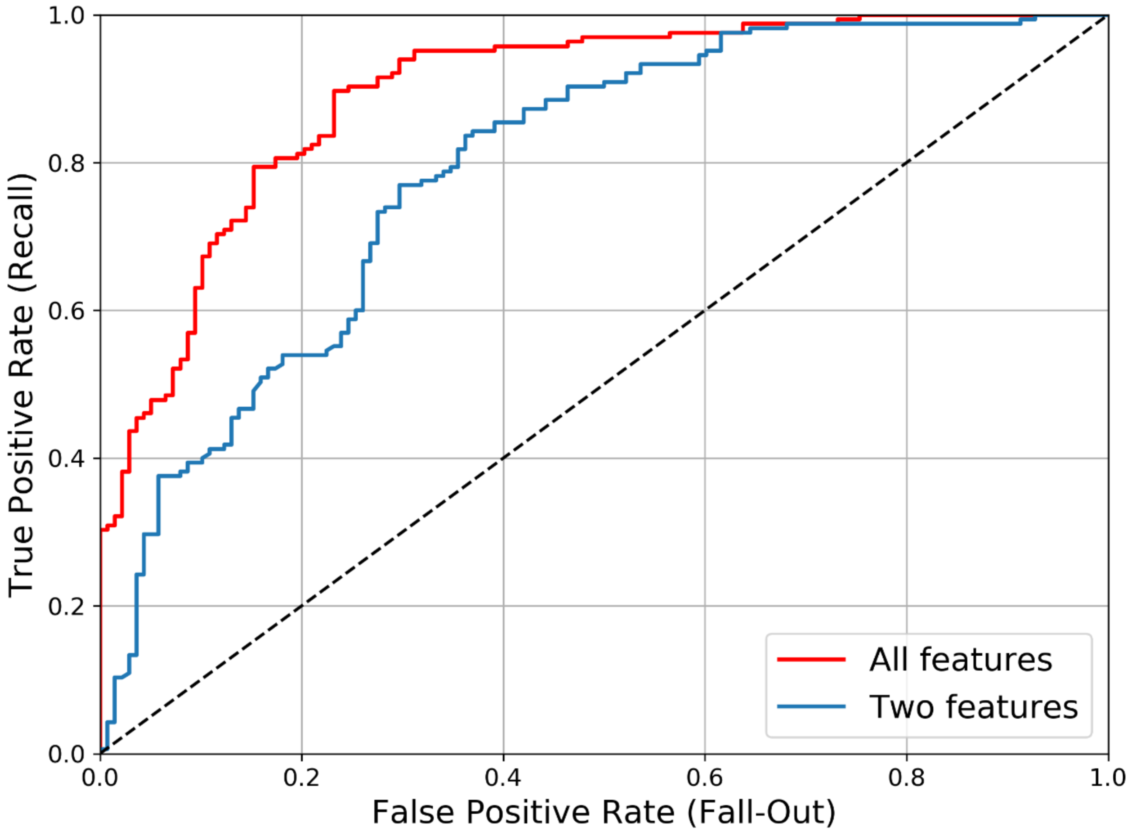
ROC curve

```
from sklearn.metrics import roc_auc_score

roc_auc_score(y, y_scores)

>> 0.7911725955204216
```



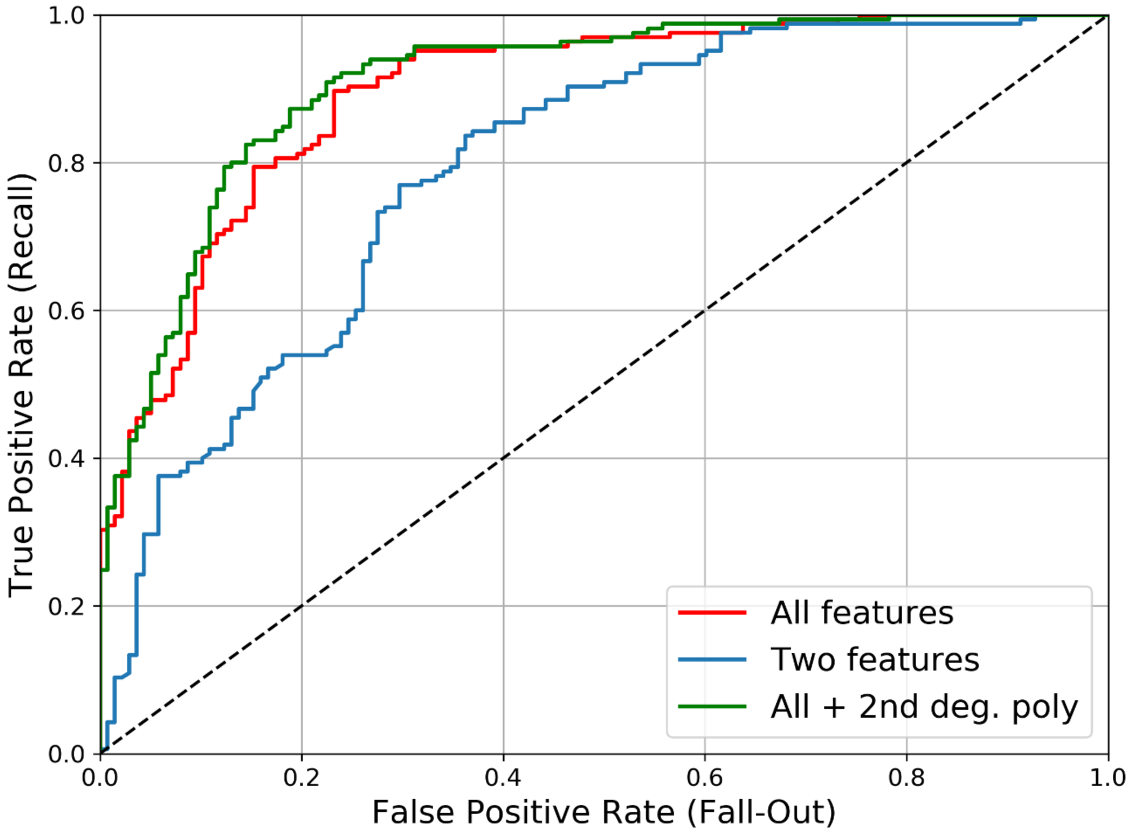
ROC curve

Logistic Regression: two features vs. all features



ROC curve

Logistic Regression: pushing harder



Work, work, work