Mudcard

- **To confirm, we do not want to perform fit_transform on the entire dataset because then information from the test/validation sets will influence how training data gets scaled?**
    - yep, fit_transform the training set, transform everything else
- **How does the array that comes from one hot allow as opposed to a unique integer? Is it just that it's unique?**
    - allow? I don't understand what that word means in this context
    - come to the office hours
- **if continuous values follow a tail distribution but are bounded, can we use minmaxscaler?**
    - I'd just go with the standard scaler
- **I'm sorry, I'm still confused about this - Can you please explain again why we need to split our dataset before pre-processing?**
    - read more about it here
- **how fit_transform do differently with transform?**
    - `fit_transform` calculates the parameters needed to transform a set (e.g., mean and stdev for the standard scaler) which is the `fit` part, then it transform the set
    - `transform` transforms a set using previously fitted parameters
    - `transform` assumes you applied `.fit` in a line above
- **I am a bit unsure about the cut offs regarding sparse datasets. Would you know of any good cut off values?**
    - I'm not sure what you mean by 'cut-off'
    - come to office hours
- **What's the importance of splitting the columns into train and test set before using onehot encoder or ordinal encoder? Why can't we transform all values all at once?**
    - different columns require different preprocessors
    - you can preprocess everything all at once using sklearn's `ColumnTransformer` as I showed at the end of the lecture
- **I'm still not sure why we shouldn't scale validation&test data. Wouldn't the ML model we made with scaled train data need to get scaled input data during validation/test process?**
    - the validation and test sets need to be transformed
    - I think that's what you mean by scaling
- **If the data is bounded but it also has a tailed distribution, is there¬† 'better' way of scaling (minmax or standard)?**
    - if in doubt, use the standard scaler!
    - that's usually good
- **What is the point of the min max scaler if we could just always use the standard scaler**

- even if you never use it (which is perfectly fine!), you might see it in other people's code so it's good to know about it
- it might come up in technical interviews: "Can you tell me the pros and cons of the standard scaler vs. the minmax scaler?"

- **I am still unclear about ignore argument in the one hot encoder**
  - work with the code in the notebook and try it without and with the ignore argument
  - print the variables if it's unclear what's going on
- **So does each column become a new feature (a new column) that is solely 0 or 1 and the original column (feature) is lost?**
  - with the one hot encoder, yes
- **For the project, will we be expected to perform the preprocessing as you showed in the first part of lecture or should we use sklearn's pipelines?**
- **In the future can we just use the scikit-learn ML pipeline functions or should we continue implementing them ourselves?**
  - feel free to use pipelines!
  - just make sure you test your code because it's easy to make mistakes with pipelines
- **Why does scaling make ml algorithms converge faster?**
  - read more here
- **When do we know when to use a transformer during preprocessing?**
  - you always need to use some transformers
  - fit_transform the training set, transform all other sets
- **I understand one-hot encoding for categoricals and how that would translate into logistic regression calculations in linear algebra, but I‚Äôm still unsure how ordinal categoricals I.e. quality [0,1,2,3,4,...] = [bad, poor, decent, good, excellent] are translated mathematically. Is it because the ordinal/hierarchy makes it more similar in nature to a continuous variable?**
  - yep, ordinal features are somewhere between categorical and continuous features
- **Didn't really see the standard scalar equation because it was hidden behind the computer screen from my vantage point on the board, which is why it was the muddiest**
  - sorry!
  - check the recording on canvas! that's a centrally placed camera
- **How does machine learning models will interpret features that are encoded by the OrdinalEncoder?**
  - if [bad, poor, decent, good, excellent] is replaced by [0,1,2,3,4], ML models will find linear correlations between the ordinal categories and the target variable

Feature selection and feature engineering

By the end of this lecture, you will be able to

- evaluate simple approaches for handling missing values

- engineer features
- select features in supervised ML

By the end of this lecture, you will be able to

- **evaluate simple approaches for handling missing values**
- engineer features
- select features in supervised ML

Dataset

- kaggle house price dataset
- check out the train.csv and the dataset description in the `data` folder!

In [1]:
```
# read the data
import pandas as pd
import numpy  as np
from sklearn.model_selection import train_test_split

# Let's load the data
df = pd.read_csv('data/train.csv')
# drop the ID
df.drop(columns=['Id'],inplace=True)

# the target variable
y = df['SalePrice']
df.drop(columns=['SalePrice'],inplace=True)
# the unprocessed feature matrix
X = df
print(X.shape)
# the feature names
ftrs = df.columns
```

```
(1460, 79)
```

In [2]:
```
print('data dimensions:',df.shape)
perc_missing_per_ftr = df.isnull().sum(axis=0)/df.shape[0]
print('fraction of missing values in features:')
print(perc_missing_per_ftr[perc_missing_per_ftr > 0])
print('data types of the features with missing values:')
print(df[perc_missing_per_ftr[perc_missing_per_ftr > 0].index].dtypes)
frac_missing = sum(df.isnull().sum(axis=1)!=0)/df.shape[0]
print('fraction of points with missing values:',frac_missing)
```

```
data dimensions: (1460, 79)
fraction of missing values in features:
LotFrontage     0.177397
Alley           0.937671
MasVnrType      0.005479
MasVnrArea      0.005479
BsmtQual        0.025342
BsmtCond        0.025342
BsmtExposure    0.026027
BsmtFinType1    0.025342
BsmtFinType2    0.026027
Electrical      0.000685
FireplaceQu     0.472603
GarageType      0.055479
GarageYrBlt     0.055479
GarageFinish    0.055479
GarageQual      0.055479
GarageCond      0.055479
PoolQC          0.995205
Fence           0.807534
MiscFeature     0.963014
dtype: float64
data types of the features with missing values:
LotFrontage     float64
Alley            object
MasVnrType       object
MasVnrArea      float64
BsmtQual         object
BsmtCond         object
BsmtExposure     object
BsmtFinType1     object
BsmtFinType2     object
Electrical       object
FireplaceQu      object
GarageType       object
GarageYrBlt     float64
GarageFinish     object
GarageQual       object
GarageCond       object
PoolQC           object
Fence            object
MiscFeature      object
dtype: object
fraction of points with missing values: 1.0
```

Simple approaches for handling missing values

- exclude points or features with missing values
- categorical feature: treat missing values as another category
- continuous feature: sklearn's SimpleImputer

Exclude points or features with missing values

- easy to do with pandas

- if missing values were encountered during data collection, it is likely missing values will occur during deployment too
    - what will you do during deployment?
    - by dropping columns/rows, you basically ignore the missing values
    - is it OK to not predict for a datapoint with missing values when the model is deployed?
        - in finance and medical problems, this is not a luxury you will have
- it's OK to temporarily drop a small fraction of rows/columns to quickly train a model and see if the project is feasible
- but if the project makes it to deployment, you will not be able to ignore the issue

Drop points or features with missing values

- not OK for the house price dataset because all points contain some NaNs.

In [3]:
```
print(df.shape)
# by default, rows/points are dropped
df_r = df.dropna()
print(df_r.shape)
# drop features with missing values
df_c = df.dropna(axis=1)
print(df_c.shape)
```

```
(1460, 79)
(0, 79)
(1460, 60)
```

Categorical feature: treat missing values as another category

- the BEST thing you can do!
- already covered in the preprocessing lecture (one hot encoding)
- example: missing values in gender
    - if survey only has options for male/female, missing values are likely because those people are outside the gender binary
    - it is a bad idea to impute (try to guess male or female and thus boxing them into the binary)
- example: native country in the adult data
    - missing data are represented as `?`
    - a one-hot encoded feature was assigned to the missing category

In [4]:
```
# read the data
import pandas as pd
import numpy  as np
from sklearn.model_selection import train_test_split

# Let's load the data
df = pd.read_csv('data/train.csv')
# drop the ID
df.drop(columns=['Id'],inplace=True)
```

```
# the target variable
y = df['SalePrice']
df.drop(columns=['SalePrice'],inplace=True)
# the unprocessed feature matrix
X = df.values
print(X.shape)
# the feature names
ftrs = df.columns
```

```
(1460, 79)
```

In [5]:
```
random_state = 42

# let's split to train, CV, and test
X_train, X_other, y_train, y_other = train_test_split(df, y, train_size=0.6, ra
X_CV, X_test, y_CV, y_test = train_test_split(X_other, y_other, test_size=0.5,

print(X_train.shape)
print(X_CV.shape)
print(X_test.shape)
```

```
(876, 79)
(292, 79)
(292, 79)
```

In [6]:
```
# collect the various features
cat_ftrs = ['MSZoning','Street','Alley','LandContour','LotConfig','Neighborhood
            'BldgType','HouseStyle','RoofStyle','RoofMatl','Exterior1st','Exter
           'Heating','CentralAir','Electrical','GarageType','PavedDrive','MiscF
ordinal_ftrs = ['LotShape','Utilities','LandSlope','ExterQual','ExterCond','Bsm
               'BsmtFinType1','BsmtFinType2','HeatingQC','KitchenQual','Functio
               'GarageQual','GarageCond','PoolQC','Fence']
ordinal_cats = [['Reg','IR1','IR2','IR3'],['AllPub','NoSewr','NoSeWa','ELO'],['
               ['Po','Fa','TA','Gd','Ex'],['Po','Fa','TA','Gd','Ex'],['NA','Po'
               ['NA','Po','Fa','TA','Gd','Ex'],['NA','No','Mn','Av','Gd'],['NA'
               ['NA','Unf','LwQ','Rec','BLQ','ALQ','GLQ'],['Po','Fa','TA','Gd',
               ['Sal','Sev','Maj2','Maj1','Mod','Min2','Min1','Typ'],['NA','Po'
               ['NA','Unf','RFn','Fin'],['NA','Po','Fa','TA','Gd','Ex'],['NA','
               ['NA','Fa','TA','Gd','Ex'],['NA','MnWw','GdWo','MnPrv','GdPrv']]
num_ftrs = ['MSSubClass','LotFrontage','LotArea','OverallQual','OverallCond','Y
             'MasVnrArea','BsmtFinSF1','BsmtFinSF2','BsmtUnfSF','TotalBsmtSF','
             'LowQualFinSF','GrLivArea','BsmtFullBath','BsmtHalfBath','FullBath
             'KitchenAbvGr','TotRmsAbvGrd','Fireplaces','GarageYrBlt','GarageCa
             'OpenPorchSF','EnclosedPorch','3SsnPorch','ScreenPorch','PoolArea'
```

In [7]:
```
# preprocess with pipeline and columntransformer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.ensemble import RandomForestRegressor

random_state = 42
```

```python
# one-hot encoder
# We need to replace the NaN with a string first!
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant',fill_value='missing')),
    ('onehot', OneHotEncoder(sparse=False,handle_unknown='ignore'))])

# ordinal encoder
# We need to replace the NaN with a string first!
ordinal_transformer = Pipeline(steps=[
    ('imputer2', SimpleImputer(strategy='constant',fill_value='NA')),
    ('ordinal', OrdinalEncoder(categories = ordinal_cats))])

# standard scaler
numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())])

# collect the transformers
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, num_ftrs),
        ('cat', categorical_transformer, cat_ftrs),
        ('ord', ordinal_transformer, ordinal_ftrs)])
```

In [8]:
```python
# fit_transform the training set
X_prep = preprocessor.fit_transform(X_train)
# the feature names after fit
feature_names = preprocessor.get_feature_names_out()

# you can convert the numpy array back to a data frame with the feature names i
df_train = pd.DataFrame(data=X_prep,columns=feature_names)
print(df_train.shape)

# transform the CV
df_CV = preprocessor.transform(X_CV)
df_CV = pd.DataFrame(data=df_CV,columns = feature_names)
print(df_CV.shape)

# transform the test
df_test = preprocessor.transform(X_test)
df_test = pd.DataFrame(data=df_test,columns = feature_names)
print(df_test.shape)
print(feature_names)
```

```
(876, 223)
(292, 223)
(292, 223)
['num__MSSubClass' 'num__LotFrontage' 'num__LotArea' 'num__OverallQual'
 'num__OverallCond' 'num__YearBuilt' 'num__YearRemodAdd' 'num__MasVnrArea'
 'num__BsmtFinSF1' 'num__BsmtFinSF2' 'num__BsmtUnfSF' 'num__TotalBsmtSF'
 'num__1stFlrSF' 'num__2ndFlrSF' 'num__LowQualFinSF' 'num__GrLivArea'
 'num__BsmtFullBath' 'num__BsmtHalfBath' 'num__FullBath' 'num__HalfBath'
 'num__BedroomAbvGr' 'num__KitchenAbvGr' 'num__TotRmsAbvGrd'
 'num__Fireplaces' 'num__GarageYrBlt' 'num__GarageCars' 'num__GarageArea'
 'num__WoodDeckSF' 'num__OpenPorchSF' 'num__EnclosedPorch'
 'num__3SsnPorch' 'num__ScreenPorch' 'num__PoolArea' 'num__MiscVal'
 'num__MoSold' 'num__YrSold' 'cat__MSZoning_C (all)' 'cat__MSZoning_FV'
 'cat__MSZoning_RH' 'cat__MSZoning_RL' 'cat__MSZoning_RM'
 'cat__Street_Grvl' 'cat__Street_Pave' 'cat__Alley_Grvl' 'cat__Alley_Pave'
 'cat__Alley_missing' 'cat__LandContour_Bnk' 'cat__LandContour_HLS'
 'cat__LandContour_Low' 'cat__LandContour_Lvl' 'cat__LotConfig_Corner'
 'cat__LotConfig_CulDSac' 'cat__LotConfig_FR2' 'cat__LotConfig_FR3'
 'cat__LotConfig_Inside' 'cat__Neighborhood_Blmngtn'
 'cat__Neighborhood_Blueste' 'cat__Neighborhood_BrDale'
 'cat__Neighborhood_BrkSide' 'cat__Neighborhood_ClearCr'
 'cat__Neighborhood_CollgCr' 'cat__Neighborhood_Crawfor'
 'cat__Neighborhood_Edwards' 'cat__Neighborhood_Gilbert'
 'cat__Neighborhood_IDOTRR' 'cat__Neighborhood_MeadowV'
 'cat__Neighborhood_Mitchel' 'cat__Neighborhood_NAmes'
 'cat__Neighborhood_NPkVill' 'cat__Neighborhood_NWAmes'
 'cat__Neighborhood_NoRidge' 'cat__Neighborhood_NridgHt'
 'cat__Neighborhood_OldTown' 'cat__Neighborhood_SWISU'
 'cat__Neighborhood_Sawyer' 'cat__Neighborhood_SawyerW'
 'cat__Neighborhood_Somerst' 'cat__Neighborhood_StoneBr'
 'cat__Neighborhood_Timber' 'cat__Neighborhood_Veenker'
 'cat__Condition1_Artery' 'cat__Condition1_Feedr' 'cat__Condition1_Norm'
 'cat__Condition1_PosA' 'cat__Condition1_PosN' 'cat__Condition1_RRAe'
 'cat__Condition1_RRAn' 'cat__Condition1_RRNe' 'cat__Condition1_RRNn'
 'cat__Condition2_Artery' 'cat__Condition2_Feedr' 'cat__Condition2_Norm'
 'cat__Condition2_PosN' 'cat__Condition2_RRAe' 'cat__Condition2_RRAn'
 'cat__BldgType_1Fam' 'cat__BldgType_2fmCon' 'cat__BldgType_Duplex'
 'cat__BldgType_Twnhs' 'cat__BldgType_TwnhsE' 'cat__HouseStyle_1.5Fin'
 'cat__HouseStyle_1.5Unf' 'cat__HouseStyle_1Story'
 'cat__HouseStyle_2.5Fin' 'cat__HouseStyle_2.5Unf'
 'cat__HouseStyle_2Story' 'cat__HouseStyle_SFoyer' 'cat__HouseStyle_SLvl'
 'cat__RoofStyle_Flat' 'cat__RoofStyle_Gable' 'cat__RoofStyle_Gambrel'
 'cat__RoofStyle_Hip' 'cat__RoofStyle_Mansard' 'cat__RoofStyle_Shed'
 'cat__RoofMatl_ClyTile' 'cat__RoofMatl_CompShg' 'cat__RoofMatl_Metal'
 'cat__RoofMatl_Roll' 'cat__RoofMatl_Tar&Grv' 'cat__RoofMatl_WdShake'
 'cat__RoofMatl_WdShngl' 'cat__Exterior1st_AsbShng'
 'cat__Exterior1st_AsphShn' 'cat__Exterior1st_BrkComm'
 'cat__Exterior1st_BrkFace' 'cat__Exterior1st_CBlock'
 'cat__Exterior1st_CemntBd' 'cat__Exterior1st_HdBoard'
 'cat__Exterior1st_MetalSd' 'cat__Exterior1st_Plywood'
 'cat__Exterior1st_Stone' 'cat__Exterior1st_Stucco'
 'cat__Exterior1st_VinylSd' 'cat__Exterior1st_Wd Sdng'
 'cat__Exterior1st_WdShing' 'cat__Exterior2nd_AsbShng'
 'cat__Exterior2nd_AsphShn' 'cat__Exterior2nd_Brk Cmn'
 'cat__Exterior2nd_BrkFace' 'cat__Exterior2nd_CBlock'
 'cat__Exterior2nd_CmentBd' 'cat__Exterior2nd_HdBoard'
 'cat__Exterior2nd_ImStucc' 'cat__Exterior2nd_MetalSd'
```

```
 'cat__Exterior2nd_Other' 'cat__Exterior2nd_Plywood'
 'cat__Exterior2nd_Stone' 'cat__Exterior2nd_Stucco'
 'cat__Exterior2nd_VinylSd' 'cat__Exterior2nd_Wd Sdng'
 'cat__Exterior2nd_Wd Shng' 'cat__MasVnrType_BrkCmn'
 'cat__MasVnrType_BrkFace' 'cat__MasVnrType_None' 'cat__MasVnrType_Stone'
 'cat__MasVnrType_missing' 'cat__Foundation_BrkTil'
 'cat__Foundation_CBlock' 'cat__Foundation_PConc' 'cat__Foundation_Slab'
 'cat__Foundation_Stone' 'cat__Foundation_Wood' 'cat__Heating_Floor'
 'cat__Heating_GasA' 'cat__Heating_GasW' 'cat__Heating_Grav'
 'cat__Heating_OthW' 'cat__Heating_Wall' 'cat__CentralAir_N'
 'cat__CentralAir_Y' 'cat__Electrical_FuseA' 'cat__Electrical_FuseF'
 'cat__Electrical_FuseP' 'cat__Electrical_SBrkr' 'cat__Electrical_missing'
 'cat__GarageType_2Types' 'cat__GarageType_Attchd'
 'cat__GarageType_Basment' 'cat__GarageType_BuiltIn'
 'cat__GarageType_CarPort' 'cat__GarageType_Detchd'
 'cat__GarageType_missing' 'cat__PavedDrive_N' 'cat__PavedDrive_P'
 'cat__PavedDrive_Y' 'cat__MiscFeature_Gar2' 'cat__MiscFeature_Shed'
 'cat__MiscFeature_TenC' 'cat__MiscFeature_missing' 'cat__SaleType_COD'
 'cat__SaleType_CWD' 'cat__SaleType_Con' 'cat__SaleType_ConLD'
 'cat__SaleType_ConLI' 'cat__SaleType_ConLw' 'cat__SaleType_New'
 'cat__SaleType_Oth' 'cat__SaleType_WD' 'cat__SaleCondition_Abnorml'
 'cat__SaleCondition_AdjLand' 'cat__SaleCondition_Alloca'
 'cat__SaleCondition_Family' 'cat__SaleCondition_Normal'
 'cat__SaleCondition_Partial' 'ord__LotShape' 'ord__Utilities'
 'ord__LandSlope' 'ord__ExterQual' 'ord__ExterCond' 'ord__BsmtQual'
 'ord__BsmtCond' 'ord__BsmtExposure' 'ord__BsmtFinType1'
 'ord__BsmtFinType2' 'ord__HeatingQC' 'ord__KitchenQual' 'ord__Functional'
 'ord__FireplaceQu' 'ord__GarageFinish' 'ord__GarageQual'
 'ord__GarageCond' 'ord__PoolQC' 'ord__Fence']
```

In [9]:

```python
print('data dimensions:',df_train.shape)
perc_missing_per_ftr = df_train.isnull().sum(axis=0)/df_train.shape[0]
print('fraction of missing values in features:')
print(perc_missing_per_ftr[perc_missing_per_ftr > 0])
print('data types of the features with missing values:')
print(df_train[perc_missing_per_ftr[perc_missing_per_ftr > 0].index].dtypes)
frac_missing = sum(df_train.isnull().sum(axis=1)!=0)/df_train.shape[0]
print('fraction of points with missing values:',frac_missing)
```

```
data dimensions: (876, 223)
fraction of missing values in features:
num__LotFrontage    0.190639
num__MasVnrArea     0.002283
num__GarageYrBlt    0.052511
dtype: float64
data types of the features with missing values:
num__LotFrontage    float64
num__MasVnrArea     float64
num__GarageYrBlt    float64
dtype: object
fraction of points with missing values: 0.23972602739726026
```

Quiz 1

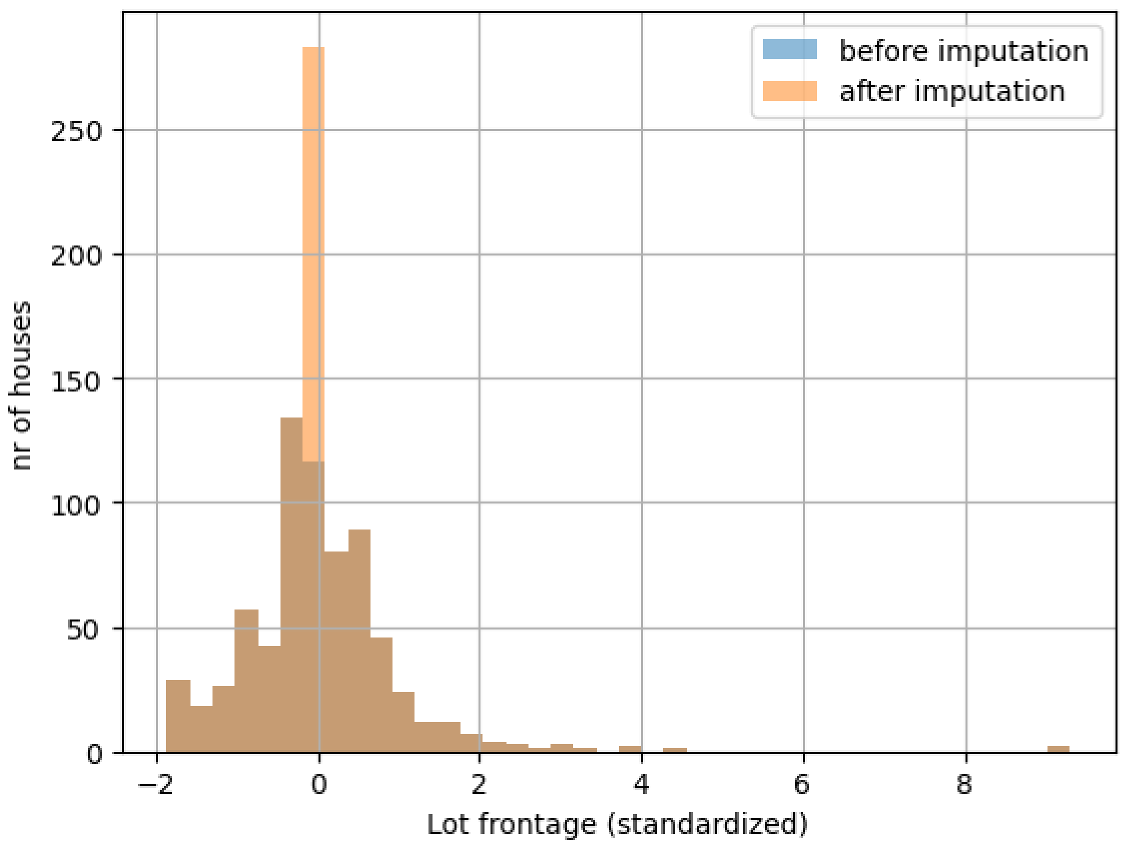
The gender feature below contains missing values. Please explain how you would encode it and would be the output of the encoder. Do not write code. The goal of this quiz is to test your conceptual understanding so write text and the output array.

gender = ['Male', 'Female', 'Male', NaN, NaN, 'Female']

Continuous feature: sklearn's SimpleImputer

- Imputation means you infer the missing values from the known part of the data
- sklearn's SimpleImputer can do mean and median imputation
- A BAD IDEA!
    - mean or median imputation decreases the variance of the feature

In [10]:
```
import matplotlib.pyplot as plt

si = SimpleImputer(strategy='mean')
X_lot = si.fit_transform(df_train[['num__LotFrontage']])

df_train['num__LotFrontage'].hist(bins=40,label = 'before imputation',alpha=0.5
plt.hist(X_lot,bins=40,label='after imputation',alpha=0.5)
plt.xlabel('Lot frontage (standardized)')
plt.ylabel('nr of houses')
plt.legend()
plt.show()

print('std before imputation:',np.std(df_train['num__LotFrontage']))
print('std after imputation:',np.std(X_lot))
```




```
std before imputation: 1.0
std after imputation: 0.8996447802291788
```

If your project dataset has missing values...

- handle missing values in categorical and ordinal features as we discussed above
- describe missing values in continuous features
  - how many continuous features contain missing values?
  - what fraction of points contain missing values?
  - what the fraction of missing values in each continuous feature?
- we will cover three advanced methods to handle missing values in continuous features in a few weeks
  - multivariate imputation
  - XGBoost
  - reduced features method (aka the pattern submodel approach)

By the end of this lecture, you will be able to

- evaluate simple approaches for handling missing values
- **engineer features**
- select features in supervised ML

Feature engineering

Automatic feature engineering:

- combine features in a simple and automatic way (PolynomialFeatures method in sklearn)
- if n_ftrs << n_points, this can modestly improve the predictive power of your model

Manual feature engineering:

- difficult, project-specific, and requires domain-knowledge
- it can boost the predictive power of your model!

Automatic feature engineering

In [11]:
```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures

X = np.arange(6).reshape(3, 2)
print(X)

poly = PolynomialFeatures(2)
print(poly.fit_transform(X)) # [1, a, b, a^2, ab, b^2]
poly = PolynomialFeatures(2, include_bias=False)
print(poly.fit_transform(X)) # [a, b, a^2, ab, b^2]
poly = PolynomialFeatures(2,interaction_only=True, include_bias=False)
print(poly.fit_transform(X)) # [a, b, ab]
```

```
[[0 1]
 [2 3]
 [4 5]]
[[ 1.  0.  1.  0.  0.  1.]
 [ 1.  2.  3.  4.  6.  9.]
 [ 1.  4.  5. 16. 20. 25.]]
[[ 0.  1.  0.  0.  1.]
 [ 2.  3.  4.  6.  9.]
 [ 4.  5. 16. 20. 25.]]
[[ 0.  1.  0.]
 [ 2.  3.  6.]
 [ 4.  5. 20.]]
```

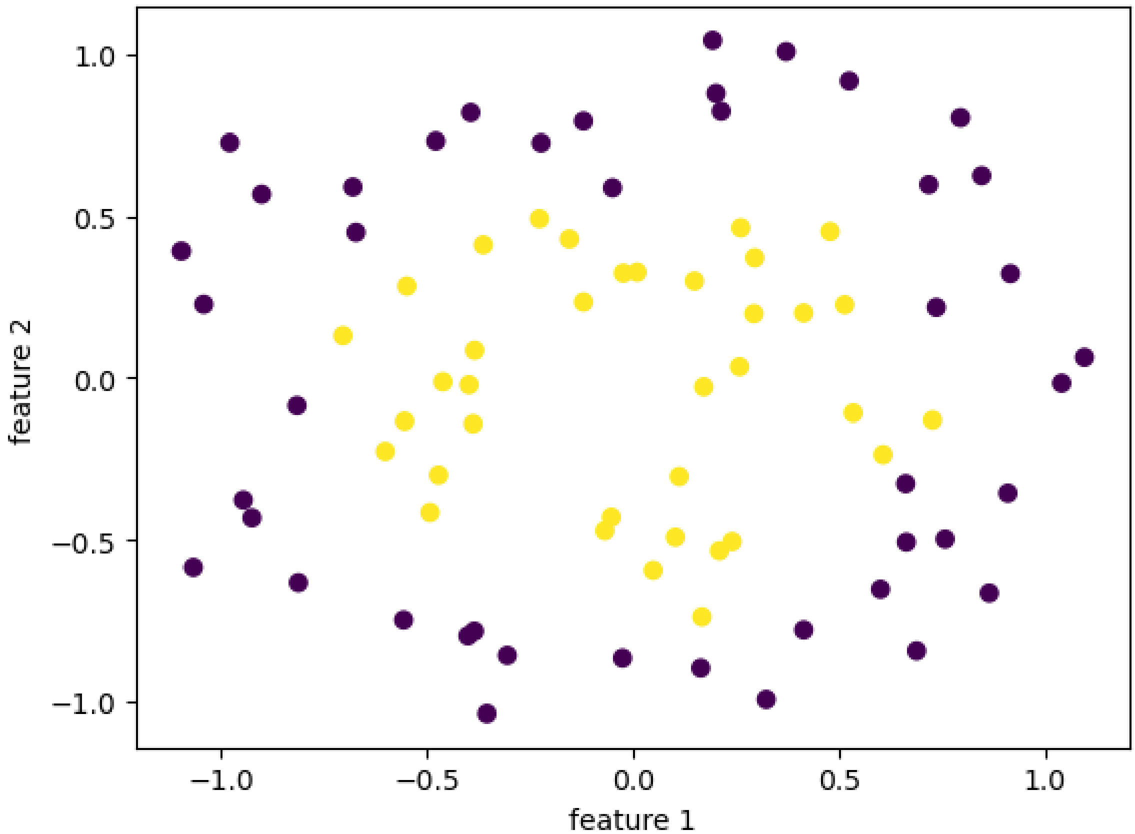
Manual feature engineering

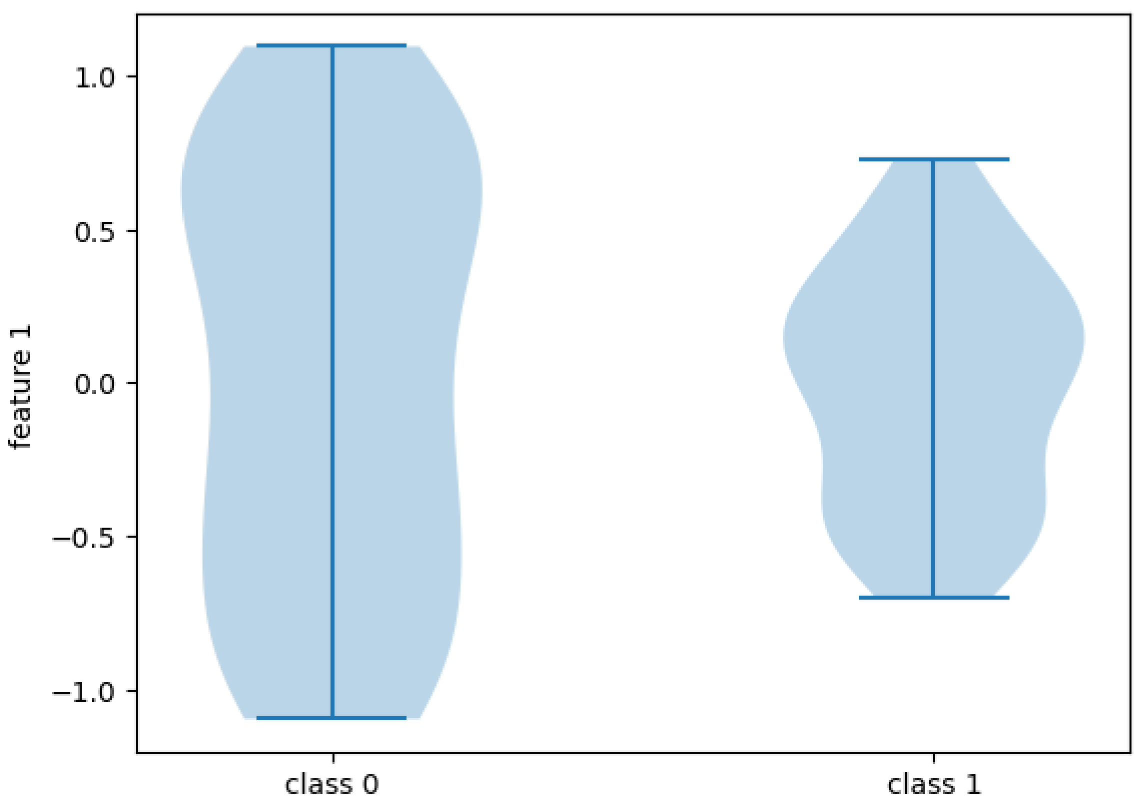
Some advice:

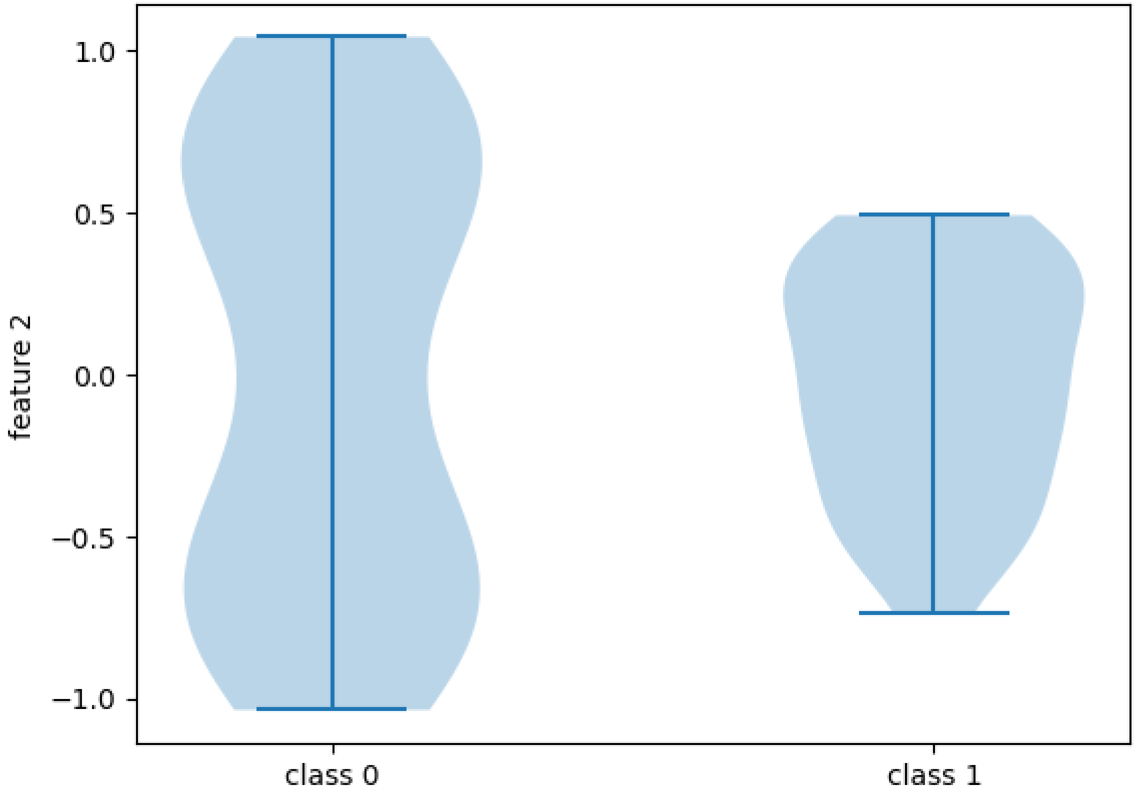
- EDA can give you insights on how you should engineer and preprocess your features better
- normalizing a feature with another feature can often be helpful
    - for example you want to predict who will attend an event
    - two features you have:
        - number of invite emails sent: [10, 20, 10, 20, 5]
        - number of email invites opened: [5, 2, 10, 10, 0]
    - a good new feature could be the fraction of invite emails opened
        - fraction of invite emails opened: [0.5, 0.1, 1, 0.5, 0]
        - person 3 might be more likely to attend than person 2 but that's only obvious from the normalized feature

In [12]:
```
from sklearn.datasets import make_circles
from sklearn.model_selection import train_test_split

X, y = make_circles(noise=0.15, factor=0.5, random_state=1)

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_st

plt.scatter(X_train[:,0],X_train[:,1],c=y_train)
plt.xlabel('feature 1')
plt.ylabel('feature 2')
plt.show()

dataset = [X_train[y_train==0,0],
           X_train[y_train==1,0]]

plt.violinplot(dataset = dataset)
plt.xticks([1,2],['class 0','class 1'])
plt.ylabel('feature 1')
plt.show()


dataset = [X_train[y_train==0,1],
           X_train[y_train==1,1]]
```

```
plt.violinplot(dataset = dataset)
plt.xticks([1,2],['class 0','class 1'])
plt.ylabel('feature 2')
plt.show()
```







In [13]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from matplotlib.colors import ListedColormap

def simple_ML_pipeline(X_train,X_test,y_train,y_test):
    LR = LogisticRegression() # logistic regression is a simple linear classifi
    LR.fit(X_train,y_train)
    y_test_pred = LR.predict(X_test)
    return accuracy_score(y_test,y_test_pred)

test_score = simple_ML_pipeline(X_train,X_test,y_train,y_test)
print(test_score)
```

```
0.3
```

In [14]:

```
# add new feature
new_feature = np.sqrt(X_train[:,0]**2+X_train[:,1]**2) # the distance from the
X_train = np.hstack((X_train,np.expand_dims(new_feature,axis=1)))
print(X_train[:5,:])
new_feature = np.sqrt(X_test[:,0]**2+X_test[:,1]**2)
X_test = np.hstack((X_test,np.expand_dims(new_feature,axis=1)))
```

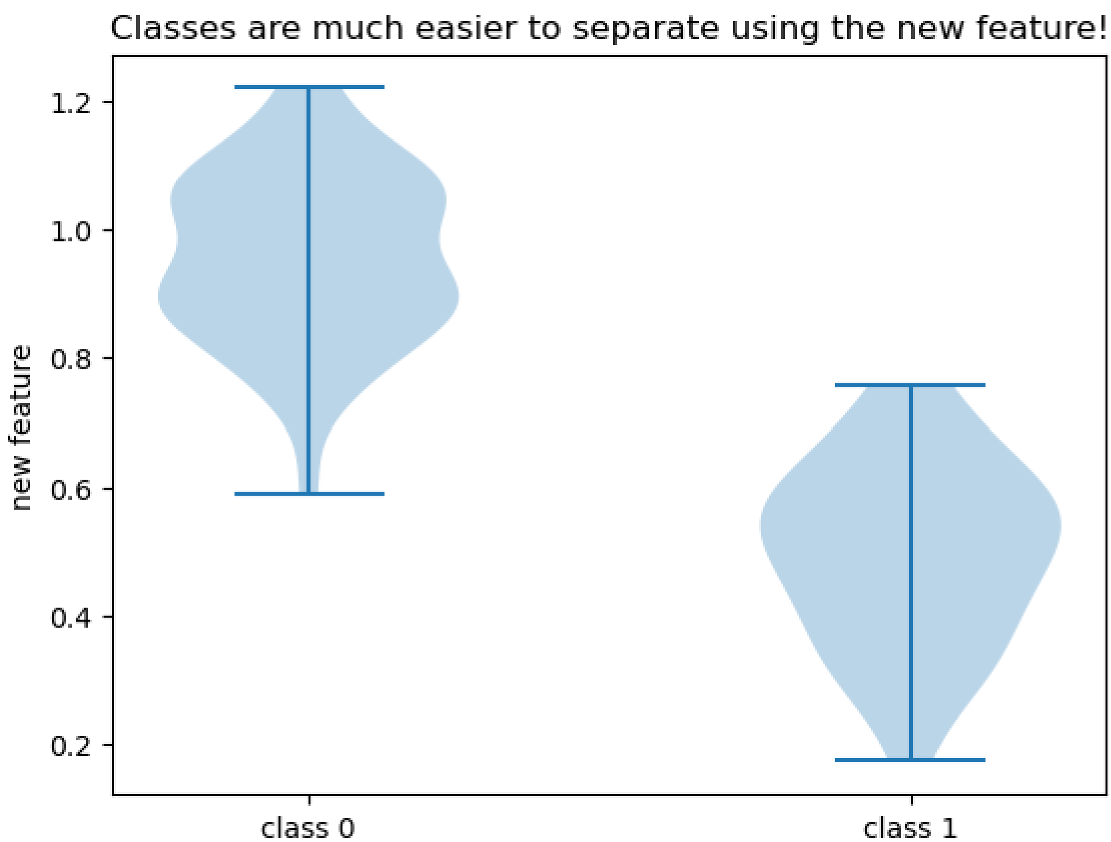
```
[[-0.05045148  0.58776084  0.58992217]
 [-0.54933449  0.28364692  0.61824264]
 [-0.55471872 -0.13344625  0.57054426]
 [-0.90194371  0.56791184  1.06584535]
 [ 0.41429957 -0.77851327  0.88188834]]
```

In [15]:

```
test_score = simple_ML_pipeline(X_train,X_test,y_train,y_test)
print(test_score) # the test accuracy improved a lot!
```

```
1.0
```

In [16]:
```
dataset = [X_train[y_train==0,2],
           X_train[y_train==1,2]]

plt.violinplot(dataset = dataset)
plt.xticks([1,2],['class 0','class 1'])
plt.ylabel('new feature')
plt.title('Classes are much easier to separate using the new feature!')
plt.show()
```



Quiz 2

X has three columns: a, b, and c.

```
X = np.arange(9).reshape(3, 3)
```

```
poly = PolynomialFeatures(degree = 2, include_bias = False)
print(poly.fit_transform(X))
```

What will be the shape of the transformed X? Do not run the code. Work the problem out with pen and paper or in your head.

By the end of this lecture, you will be able to

- evaluate simple approaches for handling missing values
- engineer features
- **select features in supervised ML**

Feature selection

We cover today how to do feature selection **before** the ML model is trained. We cover later how to select features with ML feature importances.

Necessary if

- you have too many features: n_ftrs > n_points (some algorithms break down)
- if training an ML algorithm is too computationally expensive using all the features

Approach

1. You calculate a single number metric between each feature and the target variable **using the training data only**.

- sklearn supported metrics (for both regression and classification)
    - F test (only measures linear dependency)
    - mutual information (measures non-linear dependency)
- steps:
    - do you work with a classification or regression problem?
        - regression:
            - are you interested in linear or non-linear correlations with the target variable?
                - linear: use `sklearn.feature_selection.f_regression`
                - non-linear: use `sklearn.feature_selection.mutual_info_regression`
        - classification:
            - are you interested in linear or non-linear correlations with the target variable?
                - linear: use `sklearn.feature_selection.f_classif`
                - non-linear: use `sklearn.feature_selection.mutual_info_classif`

2. Keep k best features (`sklearn.feature_selection.SelectKBest` method) or keep a certain percentile of the best features (`sklearn.feature_selection.SelectPercentile` method).

Pros:

- easy to do
- it is quicker to train ML models with fewer features

Cons:

- feature interactions are not taken into account

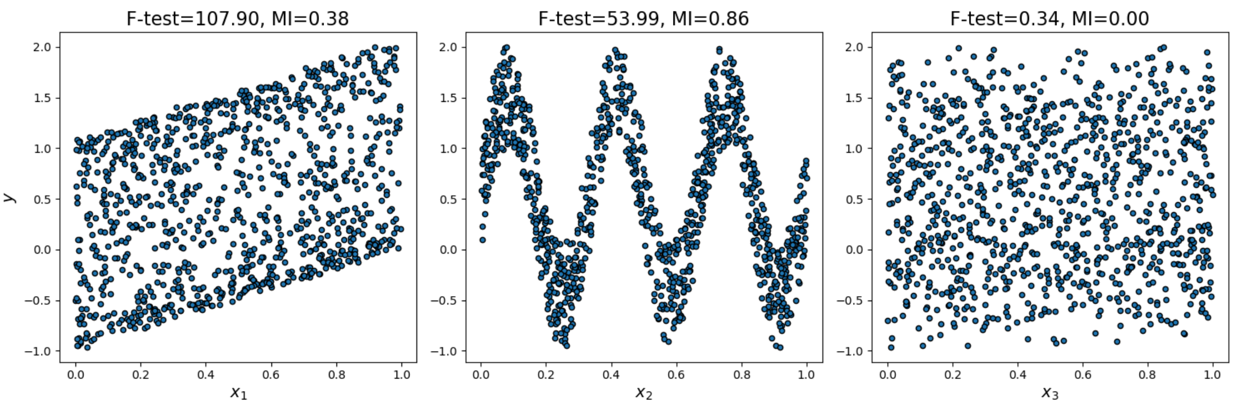
- two features separately are not predictive, but they are predictive together - such effects are ignored!

Example

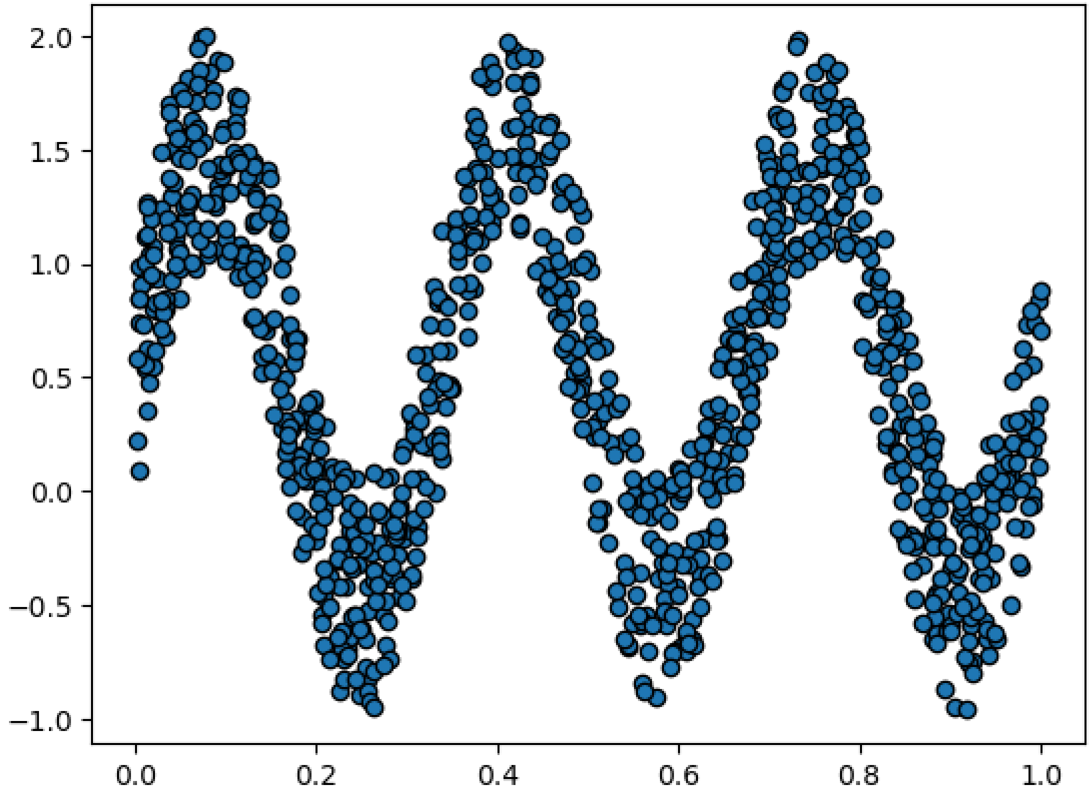
In [17]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_selection import f_regression, mutual_info_regression
np.random.seed(10)

X = np.random.rand(1000,3)
y = X[:,0] + np.sin(6 * np.pi * X[:,1]) + 0.1 * X[:,2]

f_test, p_values = f_regression(X, y)
print('f score',f_test)
print('p values',p_values)
mi = mutual_info_regression(X, y)
print('mi',mi)
```

```
f score [107.90134156  53.99212018   0.34354216]
p values [4.52216746e-24 4.18146945e-13 5.57924253e-01]
mi [0.37637501 0.86317726 0.        ]
```

In [18]:
```
plt.figure(figsize=(15, 5))
for i in range(3):
    plt.subplot(1, 3, i + 1)
    plt.scatter(X[:, i], y, edgecolor='black', s=20)
    plt.xlabel("$x_{}$".format(i + 1), fontsize=14)
    if i == 0:
        plt.ylabel("$y$", fontsize=14)
    plt.title("F-test={:.2f}, MI={:.2f}".format(f_test[i], mi[i]),
              fontsize=16)
plt.tight_layout()
plt.show()
```



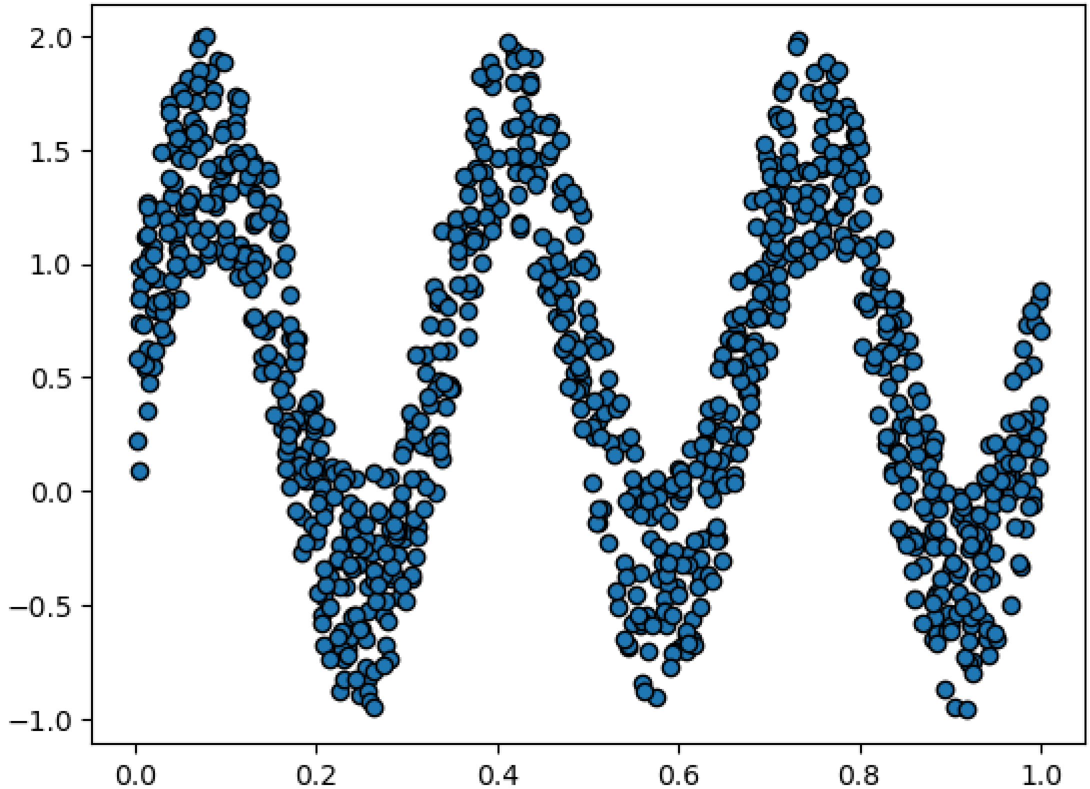
In [19]:
```
from sklearn.feature_selection import SelectKBest
f_select = SelectKBest(mutual_info_regression,k=1)
X_f = f_select.fit_transform(X,y)
plt.scatter(X_f,y,edgecolor='k')
plt.show()
# the features selected:
print(f_select.get_support())
```




```
[False  True False]
```

In [20]:

```
from sklearn.feature_selection import SelectPercentile
f_selector = SelectPercentile(mutual_info_regression,percentile=33)
X_mi = f_selector.fit_transform(X,y)
plt.scatter(X_mi,y,edgecolor='k')
plt.show()
# features selected
f_selector.get_support()
```



Out[20]: array([False, True, False])

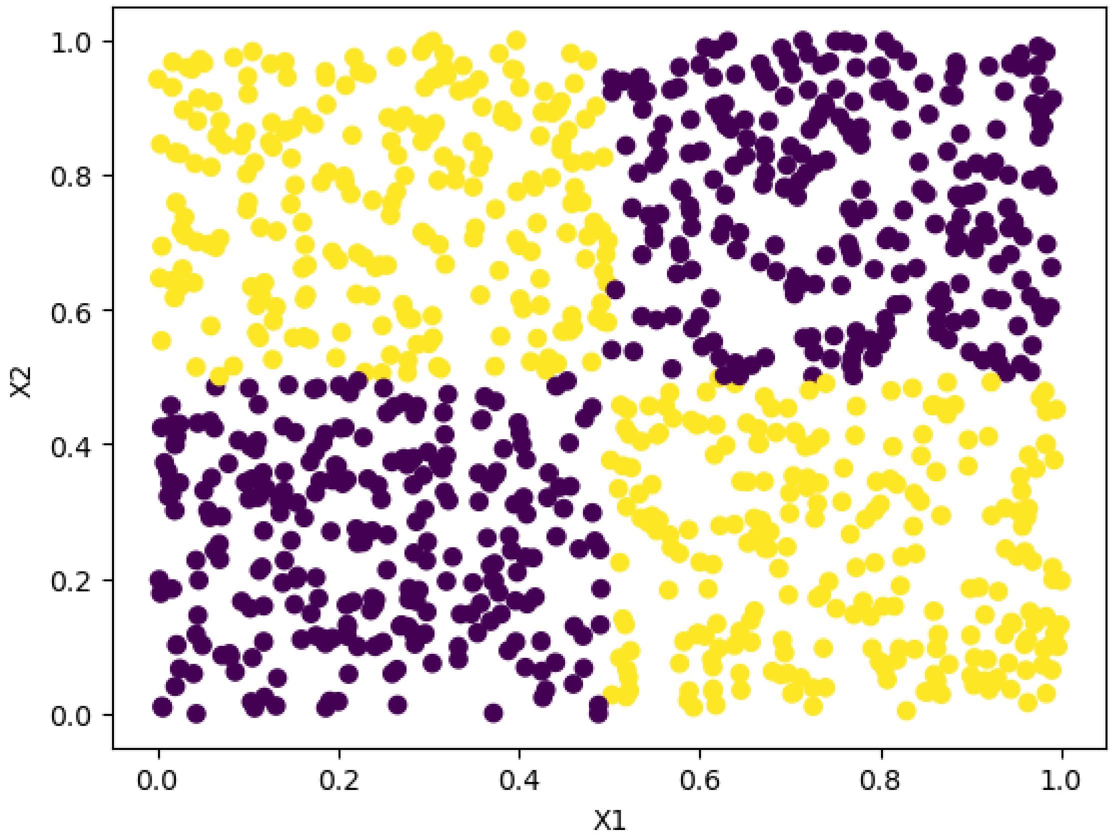
Be careful though!

In [21]:
```
# toy data
import pandas as pd
import numpy as np
from sklearn.feature_selection import f_classif, mutual_info_classif
np.random.seed(0)

X = np.random.uniform(size=(1000,2))

y = np.zeros(1000)
y[(X[:,0]>=0.5)&(X[:,1]<0.5)] = 1
y[(X[:,0]<=0.5)&(X[:,1]>0.5)] = 1
```

In [22]:
```
f_test, p_values = f_classif(X, y)
print('f score',f_test)
print('p values',p_values)

mi = mutual_info_classif(X, y)
print('mi',mi)
```

```
f score [0.28282382 0.82026181]
p values [0.59497468 0.36532223]
mi [0.00338502 0.00055867]
```

In [23]:
```
plt.scatter(X[:,0],X[:,1],c=y)
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```



Mudcard