**Linux**



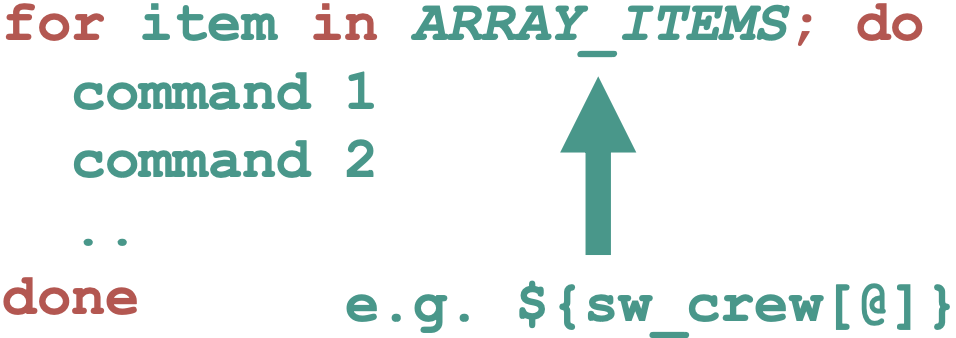
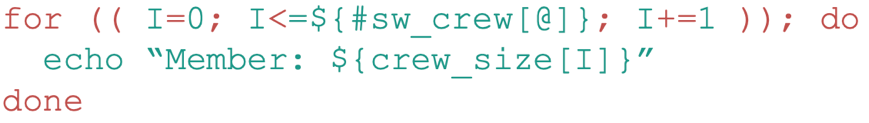
chown user:group path/to/file\_or\_directory

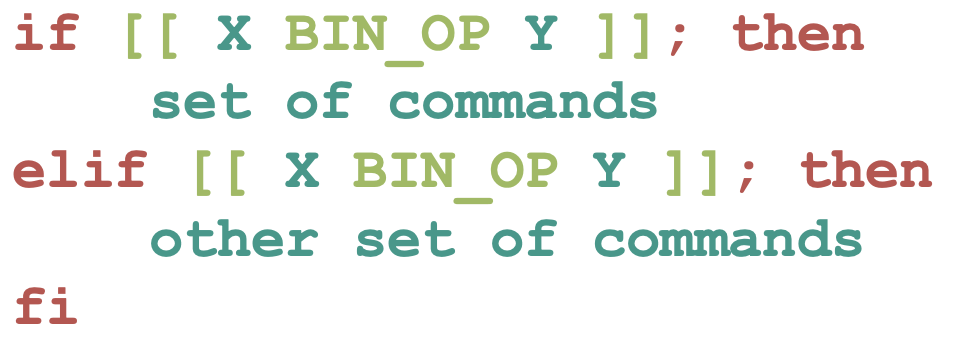
chown -R user path/to/directory

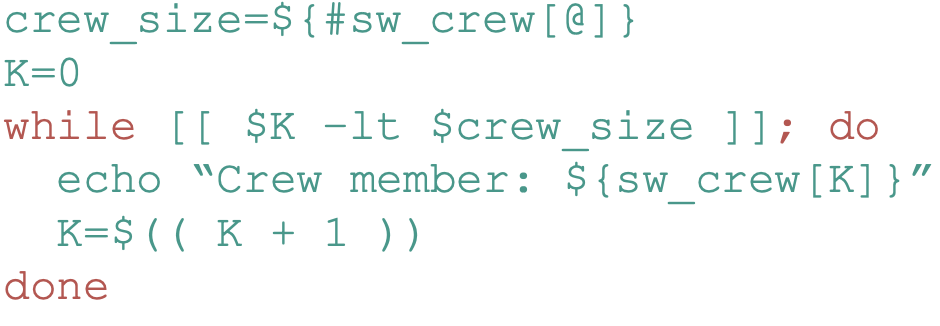
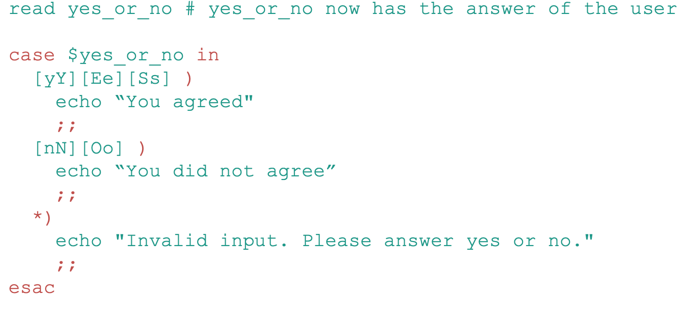
chgrp group path/to/file\_or\_directory

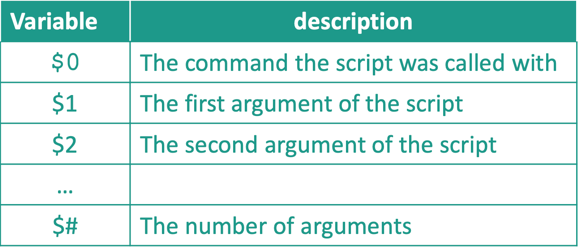
chgrp -R group path/to/directory

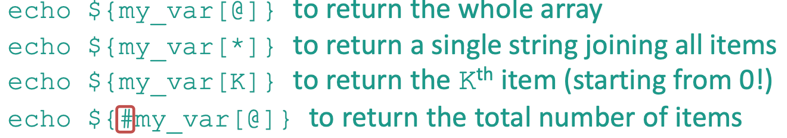
my\_files=(/path/to/files/\*.csv)

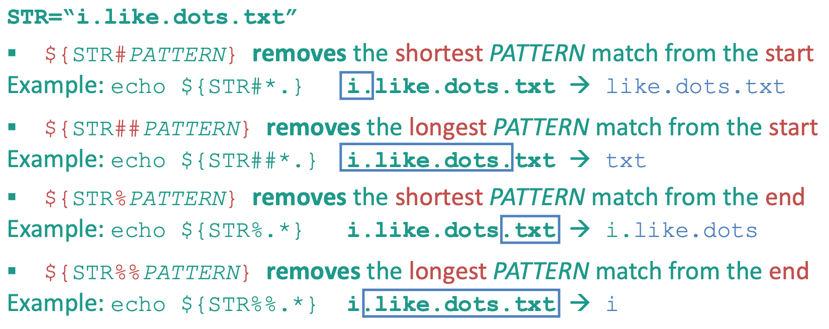
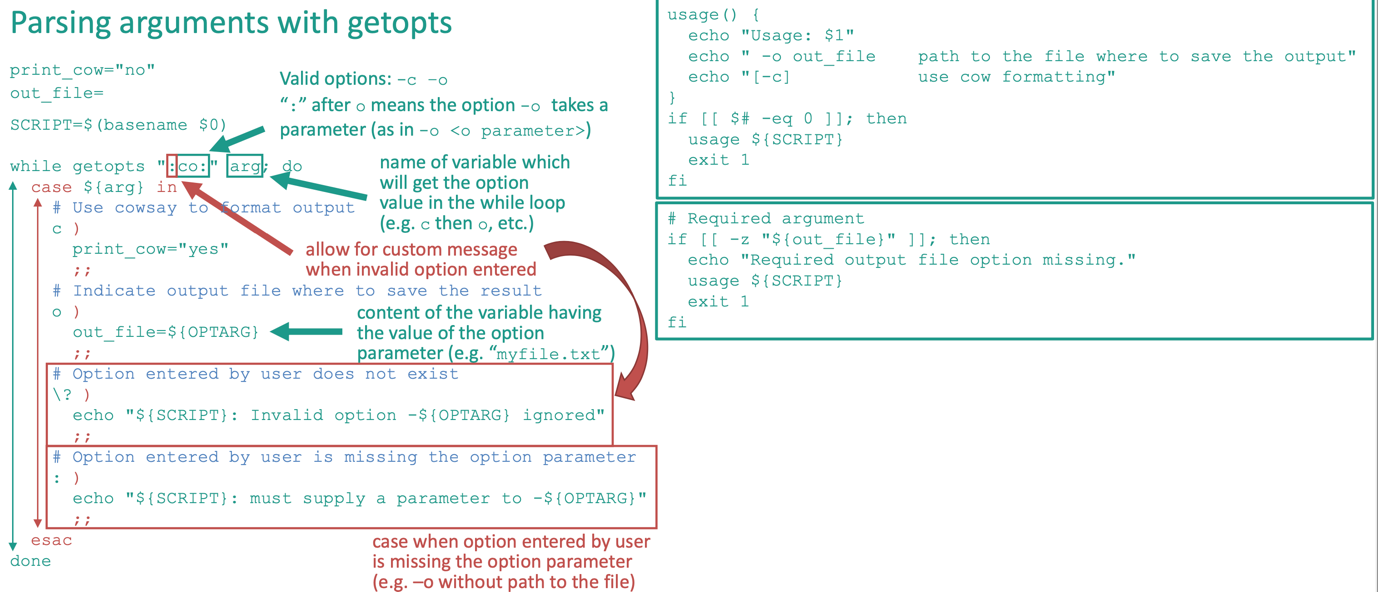












IP + Port = Socket (52.212.137.39:8080)



ssh-keygen -t rsa -b 4096

ssh-copy-id username@remote\_host

**Git**

* man git-command (e.g. man git-commit)
* git init
* git add .
* git commit -m “message”
* commit includes checksum, previous checksum, message, author, and date
* git status
* git log (--oneline OR –summary)
* git show
* git diff OldCommit NewCommit
* git difftool OldCommit NewCommit
* gitk
* git config --global user.name “NAME”
* git config --global user.email “email”

*Branches*

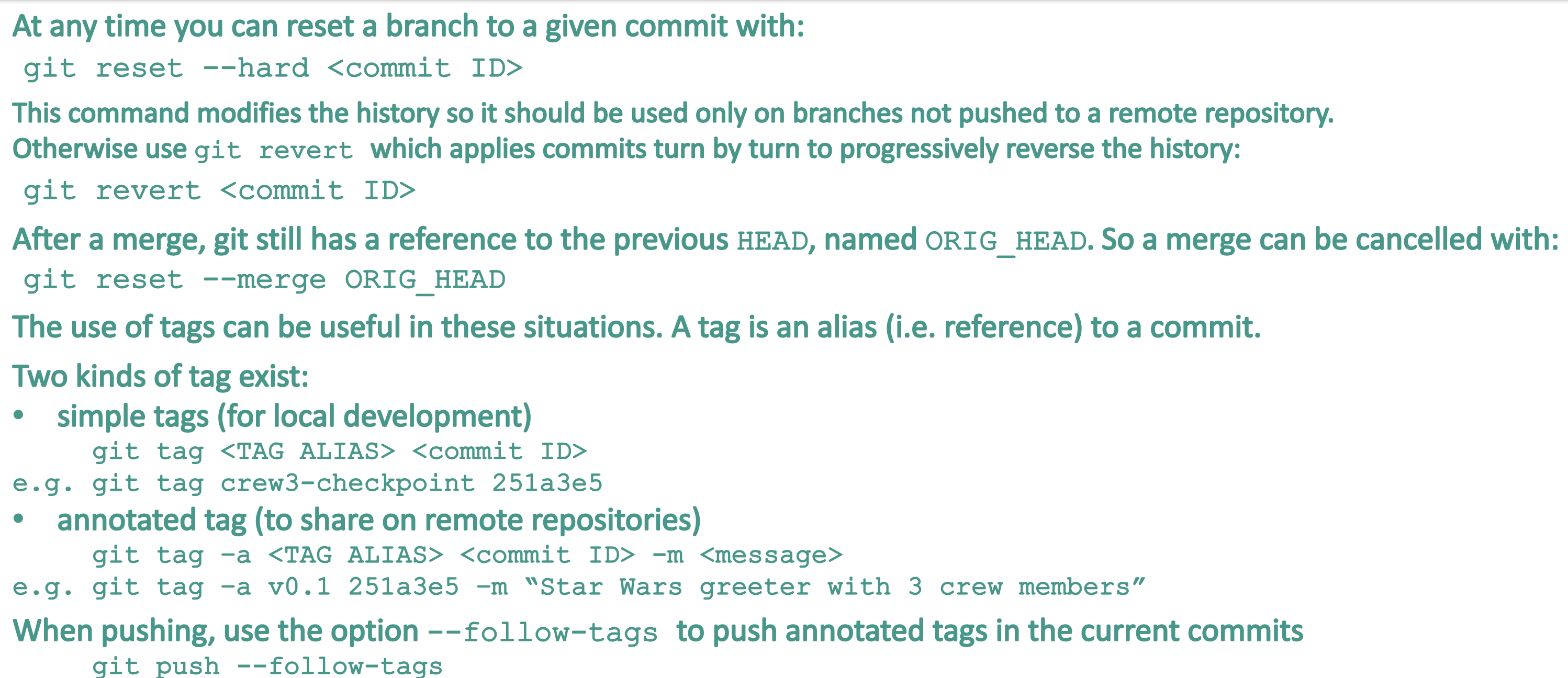
* git branch topic
* git checkout topic
* git branch topic <COMMIT-id> 🡪 creates the topic branch at COMMIT-id
* branch is a pointer to a commit
* HEAD is a pointer to the active branch
* git branch -v 🡪 lists all the branches
* git branch -d topic 🡪 deletes the branch
* git branch -M main 🡪 renames the branch

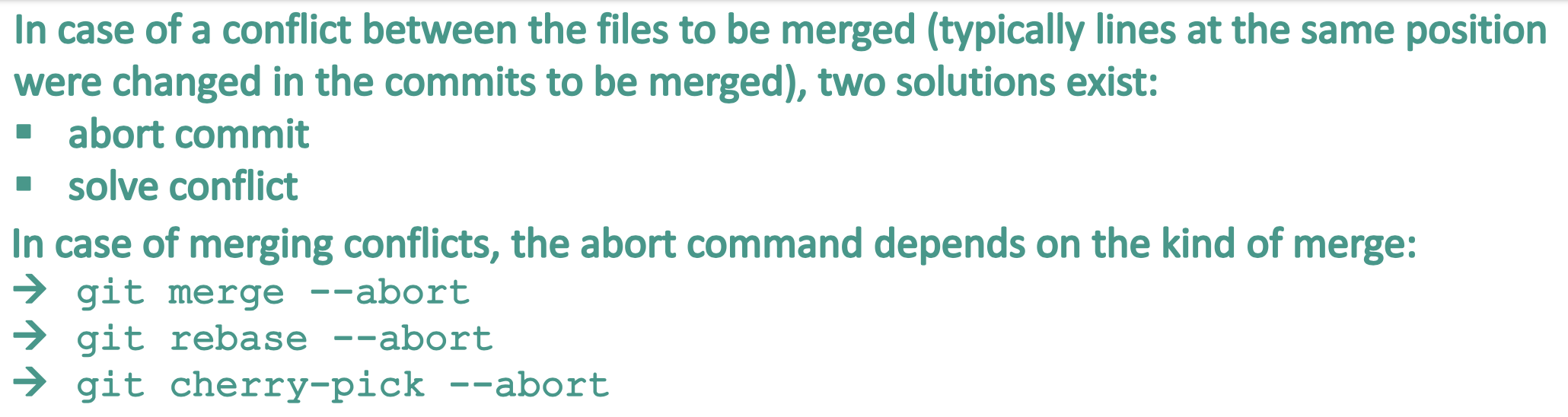
*Remote*

* git remote add origin [git@github.com:<user>/<repo>.git](mailto:git@github.com:%3cuser%3e/%3crepo%3e.git)
* git clone
* git push -u origin main
* git pull origin main (same as: git fetch origin main + git merge origin/main)
* git branch -avv (shows all local and remote branches)
* git branch -u origin/main (restores the remote tracking in the now github repo)

*Merge*

* If main branch is paused:
  + git checkout main
  + git merge topic (we can use --no-ff here to keep the history)
* If 3 way:
  + git checkout topic
  + git rebase main
  + git checkout main
  + git merge topic --no-ff
* If 3 way:
* git checkout main
* git cherry-pick <commit-id> (or <commit-id1>~..<commit-id2>)
* first commit-id is ignored.
* ~ means one before





If we want to solve the conflict, we can use “git mergetool”

**Python**

def comment\_grade(grade: int, mode: str = 'normal') -> str :

“””

Provide a feedback according to the grade value

Parameters

----------

grade

The grade obtained by the student (out of 10)

mode

The feedback mode, either "normal" (default) or "positive\_reinforcement"

Returns

-------

comment

The grade feedback

Examples

-------------

>>> comment\_grade(6)

‘Grade high enough’

“””

If1 grade >= 0 and grade < 5:

return('Grade too low')

elif1 grade > 5 and grade <= 10:

if2 mode == 'normal':

return('Grade high enough')

elif2 mode == 'positive\_reinforcement':

return('Well done, keep going!')

else2:

raise ValueError('The mode should be "normal" or "positive\_reinforcement"')

else1:

assert (grade < 0 or grade > 10), 'INTERNAL BUG: grade is not less than 0 or greater than 10'

raise ValueError('EXTERNAL ERROR: The grade entered should be between 0 and 10')

* os.getcwd()
* sys.path
* os.\_\_file\_\_
* if \_\_name\_\_==”\_\_main\_\_”: (if executed from *command line*)
* import importlib; importlib.reload(numpy)
* def test\_grade():

assert comment\_grade(0, “normal”) == ”Grade too low”

* pytest --doctest-modules file.py
* class Student:

def \_\_init\_\_(self, first\_name, last\_name, grade=None):

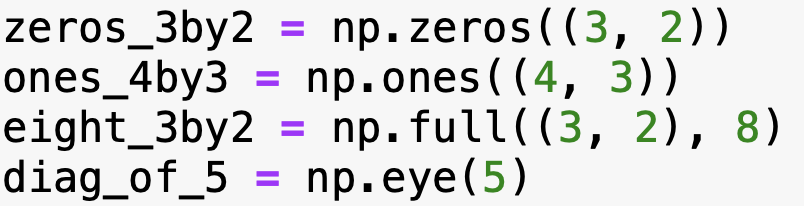
self.first\_name=first\_name

self.last\_name=last\_name

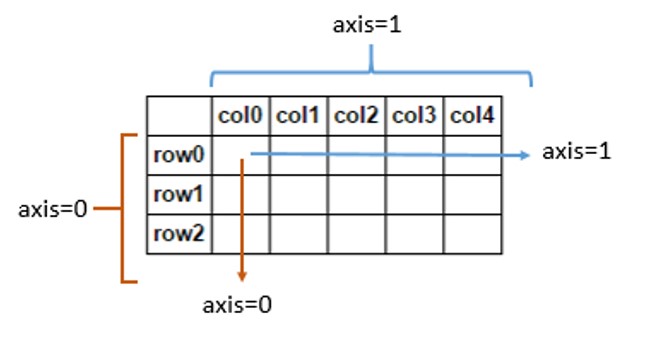
self.grade=grade

def get\_name():

return self.first\_name + “ “ + self.last\_name

* docstring for the class has Attributes and Methods
* student1 = Student(Ali, Saadat, 20)
* 
* Reading files:
  + home\_path=os.path.expanduser(‘~’)
  + file\_path=os.path.join(home\_path, “Documents”, “data”, “file.csv”)
  + with open(file\_path, ‘r’) as file\_handler:

file\_content=file\_handler.read()

* all\_files=sort(glob.glob(“~/data/\*.csv”))
* patient\_files=[f for f in all\_files if not “small” in f]
* np.loadtext(fname=”/path/to/file”, delimeter=’,’)
* 
* df = pd.readcsv(“/path/to/file”)
  + df.describe()
  + df.head(5)
  + df.shape
  + df.dtypes
  + df.isna().sum()
  + df[ df[‘year]>=2000 ]
  + df.dropna()
  + df[‘country’].value\_counts()
  + top10 = list(df[‘country’].value\_counts().nlargest(10).index)

top10\_mask=df[‘country’].isin(top10)

df[top10\_mask]

* + df[‘country’].value\_counts().nlargest(10).plot(kind=’bar’)
* sns
  + sns.boxplot(df[‘year’])
  + sns.histplot(df[‘year’])
  + sns.kdeplot(df[‘year’], df[‘price’])
  + sns.scatterplot(x=’year’, y=’price’, data=df)
  + sns.countplot(y=’country’, data=df)
  + sns.violinplot(x=”price”, y=”country”, data=df)
  + with sns.plotting\_context("notebook", font\_scale=1.2):

g = sns.catplot(x="model", y="val", hue="stage", col="scorer",

data=train\_test\_results\_df, kind="bar", sharey=False)

**ML**

Train\_test\_split

* from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=42)

LinearRegression, MSE, R2

* from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

lm = LinearRegression()

lm.fit(X\_train, y\_train)

y\_pred = lm.predict(X\_train)

R2\_train = r2\_score(y\_train, y\_pred)

MSE\_train = mean\_squared\_error(y\_train, y\_pred)

Pipeline

* from sklearn.preprocessing import PolynomialFeatures

from sklearn.pipeline import Pipeline

lm\_deg5 = Pipeline([('poly\_transformer', PolynomialFeatures(degree=5)),

('lm', LinearRegression())])

lm\_deg5.fit(X\_train, y\_train)

lm\_deg5['lm'].coef\_

y\_pred = lm\_deg5.predict(X\_train)

CrossValidation

* from sklearn.model\_selection import KFold

ml\_models = {'lm': LinearRegression(),

'lm\_deg2': Pipeline([('poly\_transformer', PolynomialFeatures(degree=2)),

('lm', LinearRegression())]),

'lm\_deg5': Pipeline([('poly\_transformer', PolynomialFeatures(degree=5)),

('lm', LinearRegression())])}

kf\_results = []

kfs = KFold(n\_splits=10, shuffle=True, random\_state=42)

for i\_f, (ix\_train, ix\_test) in enumerate(kfs.split(X\_train)):

# Loop over models

for mod\_name, mod in ml\_models.items():

# Define training and testing folds

X\_training\_folds = X\_train.iloc[ix\_train]

y\_training\_folds = y\_train.iloc[ix\_train]

X\_test\_fold = X\_train.iloc[ix\_test]

y\_test\_fold = y\_train.iloc[ix\_test]

# Fit the model on the training folds

mod.fit(X\_training\_folds, y\_training\_folds)

# Test on both the training and testing folds to check for over-/under-fitting

y\_pred\_train = mod.predict(X\_training\_folds)

y\_pred\_test = mod.predict(X\_test\_fold)

# R2

kf\_results.append({'model': mod\_name, 'fold': i\_f, 'stage': 'train', 'scorer': 'r2',

CrossValidation automated

* from sklearn.model\_selection import cross\_val\_score

from sklearn.metrics import fbeta\_score, make\_scorer

mse\_scorer = make\_scorer(mean\_squared\_error, greater\_is\_better=False)

cv\_test\_scores = {}

for mod\_name in ml\_models.keys():

cv\_test\_scores[mod\_name] = cross\_val\_score(ml\_models[mod\_name], X\_train, y\_train,

cv=kfs, scoring=mse\_scorer, n\_jobs=-1)

cv\_test\_scores\_df = pd.DataFrame(cv\_test\_scores)

CrossValidation aturomated2

* from sklearn.model\_selection import cross\_validate

cv\_scores = {}

for mod\_name in ml\_models.keys():

cv\_scores[mod\_name] = cross\_validate(ml\_models[mod\_name], X\_train, y\_train, cv=kfs,

scoring=['r2', 'neg\_mean\_squared\_error'],

return\_train\_score=True, n\_jobs=-1)

def crossval\_to\_df(cv\_dict):

crossval\_results = []

for model in cv\_dict.keys():

for scorer in cv\_dict[model].keys():

if scorer.startswith('train\_'):

score = scorer.replace('train\_', '')

for i\_val, val in enumerate(cv\_dict[model][scorer]):

crossval\_results.append({'model': model, 'fold': i\_val, 'stage': 'train',

'scorer': score, 'val': val})

elif scorer.startswith('test\_'):

score = scorer.replace('test\_', '')

for i\_val, val in enumerate(cv\_dict[model][scorer]):

crossval\_results.append({'model': model, 'fold': i\_val, 'stage': 'test',

'scorer': score, 'val': val})

return pd.DataFrame(crossval\_results)

crossval\_df = crossval\_to\_df(cv\_scores)

Transform Label and Features

* from sklearn.preprocessing import LabelEncoder, OrdinalEncoder, OneHotEncoder, StandardScaler

from sklearn.compose import make\_column\_transformer

y\_num = LabelEncoder().fit\_transform(y\_cat)

cols\_ordinal = ["smoking\_status"]

cols\_non\_ordinal = ["gender", "ever\_married", "work\_type", "Residence\_type"]

cols\_continuous = ["age", "avg\_glucose\_level", "bmi"]

preprocessor = make\_column\_transformer(

(OrdinalEncoder(categories=[['never smoked', 'formerly smoked', 'smokes']]), cols\_ordinal),

(OneHotEncoder(drop='if\_binary', sparse=False), cols\_non\_ordinal),

(StandardScaler(), cols\_continuous),

remainder='passthrough',

verbose\_feature\_names\_out=False

)

X\_final = pd.DataFrame(data=preprocessor.fit\_transform(X),

columns=preprocessor.get\_feature\_names\_out())

Logistic Regression

* from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression(penalty='none', class\_weight='balanced', max\_iter=1000)

lr.fit(X\_train, y\_train)

Confusion Matrix

* from sklearn.metrics import accuracy\_score, confusion\_matrix, ConfusionMatrixDisplay

y\_pred\_train = lr.predict(X\_train)

accuracy\_score(y\_train, y\_pred\_train)

confusion\_matrix(y\_train, y\_pred\_train)

ConfusionMatrixDisplay.from\_estimator(lr, X\_train, y\_train, display\_labels=label\_encoder.classes\_,

normalize='true')

lr\_coefs\_df = pd.DataFrame({'coefs': np.std(X\_train, 0)\*lr.coef\_[0]},

index=X.columns)

plt.figure(figsize=(6,6))

sns.barplot(x='coefs', y=lr\_coefs\_df.index, data=lr\_coefs\_df)

PCA

* from sklearn.datasets import load\_digits

from sklearn.decomposition import PCA

digits = load\_digits()

pca = PCA(n\_components=2)

projected\_digits = pca.fit\_transform(digits.data)

pca = PCA().fit(digits.data)

plt.plot(np.cumsum(pca.explained\_variance\_ratio\_))

Kmeans

* from sklearn.cluster import KMeans

from sklearn.preprocessing import scale

X = scale(digits.data)

y = digits.target

X\_pca = PCA(n\_components=2).fit\_transform(X)

kmeans = KMeans(n\_clusters=n\_digits)

kmeans.fit(X\_pca)

silhouette

* from sklearn.metrics import silhouette\_samples, silhouette\_score

clusterer = KMeans(n\_clusters=n\_clusters, random\_state=10)

cluster\_labels = clusterer.fit\_predict(X\_pca)

silhouette\_avg = silhouette\_score(X\_pca, cluster\_labels)

sample\_silhouette\_values = silhouette\_samples(X\_pca, cluster\_labels)