Data Scientist Nanodegree Program

2. Introduction to Data Science
Lesson 1: Data Science Process

CRISP-DM - Cross-Industry Standard Process for Data Mining
- business understanding
- data understanding - data collection, sorting, cleaning
- prepare data
- data modeling
- evaluate results
- deploy

→ Data → Question → Solution data exploration
Question → Data → Solution data discovery

Descriptive analytics - basic statistics to answer basic questions
- e.g. percent,
histograms
mean, max, min
grouping ...

Data modeling - predictions.
e.g. machine learning

- Points of concern for passing data to ML models in sklearn:
1. missing values
2. categorical values

3 strategies:
1. remove or drop rows with missing values
2. impute missing values
3. build models around them

- Removing data can lead to bias in model.
  Track missing values using indicator values or counts.

  ok to drop missing data
  - errors
  - not relevant
  - no variability, or constant values

  Drop column
  - significant % of rows missing values
  - replace with indicator of missing or present values (e.g. 1 for present, 0 for missing)

  Imputing
  - most common way to fill missing values
    - mean, median, mode
    - ML and bayesian statistical approaches
      K-nearest neighbors
      AMELIA library

  - introduces Bias
    - lead to overgeneralizat
    - dilute importance of feature, by reducing variability of feature and thereby power of predicti

  - common -
    - impute mean of numerical
    - impute mode of categorical
    - use KNN
    - impute either a very small value, or very large number to differentiate missing values.

- Categorial variables - convert to numbers
  - easy
  - flexible
  - scaling issues
  - added columns may lead to inclusive solutions

  } advantages

  } disadvantages

pandas.get_dummies(df, prefix,
prefix_sep = '_',
drop_first = True,
dummy_na = True)

Overfitting - model does not generalize to data not seen before.

Cold start problem - recommender engine with no prior data

Steps
1. how many missing values
2. remove data
3. impute data
4. categorial to numerical
5. different cut offs for number of non-zero values / missing values vary and find optimum outcome per cut off.
   determine how much variability is needed.

Git
- git clone
- git add *
- git commit -m 'commit here'
- git push

6. separate predictors and response
7. split into train + test data set
8. fit to model
9. score and evaluate.

3. Software Engineering

Lesson 1-5: Introduction to Software Engineering

Lesson 6: Web Development

- Basics of web app
  - html - content
  - css - style, color layout
  - javascript - interactivity, animations
- Front-end libraries
  - bootstrap
  - plotly - visualization
- Back-end libraries
  - flask
- Deployment

server ⟷ php, python, java, ruby ⟷ html, css, javascript

database ↗

Example HTML

```
<html>
<head>
   <title>    </title>
</head>

<body>
   <h1> Example HTML </h1>
   <a href = "https://www.google.com"> Home Page </a>
   <p>        </p>
   <img src = "photo.jpg" alt = "photo">
</body>
</html>
```

```
<div>
  <p> This is a <span> text </span> </p>
</div>
```

ID - name for each tag, used one time only
- e.g. `<p id = 'first_id'> </p>`
`<div id = 'first_div_id'> </div>`

class - class name reusable for multiple tags for same styling.
- multiple classes together
- e.g `<p class = 'class_one' 'class_two'> </p>`

writing css
- inline
`<p style = "font-size: 20px;"> Text </p>`

- stylesheet within HTML
```
<head>
  <style>
    p { font-size: 20 px; }
  </style>
</head>
```

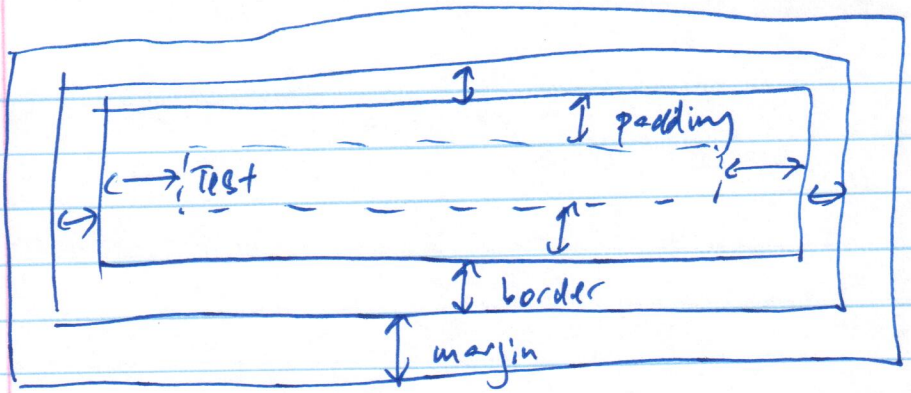
- separate .css file
```
<head>
  <link rel = "stylesheet" type = "text/css" href = "style.css">
</head>
```

```
p {
  color: red;
}
```

```
a {
  text-decoration: none;
  color: blue;
  font-weight: bold;
}
```

```
.class_name {
}
```

```
#id_name {
}
```



Specifying sizes:

1. px
2. percentage
3. em (default font size)

Bootstrap library

- class = "img-fluid"
- div class = "col-x"
  div class = "mt-x" "mb-x"
  "ml-x" "mr-x"
  h4 class = "text-muted"

Javascript

- jQuery - Javascript library

```
<script src = "https://ajax.googleapis.com/.... /jquery>
</script>
<script>
    $(document).ready(function () {
        $("img").click(function () {
            $("h1").text("Replace text");
    });
});
</script>
```

- high level language - python, PHP, Ruby, C++
- newer Javascript tools - React, Angular for front end
  - node.js for backend

ready(function () { }) - waits for html document to load
$("img") - element
$("p#first-id") - paragraph element with id = first-id

Plotly:
<script src = "https://plotly ..."></script>

```
<body>
  <div id: "plot1"></div>
  <script src = "plot1.js"></script>
</body>
```

plot1.js -

```
var xdata
var ydata
var trace1 = {
      x: xdata,
      y: ydata,
      mode: 'lines',
      type: 'scatter'
  };
var data = [trace1];
var layout = {
  title: '                ',
  xaxis: {
    title: '  ',
    tick0:  ,
    dtick:
  }
  yaxis: {
    title: "      "
  }
Plotly.newPlot('plot1', data, layout);
```

Chart library
- d3.js
- chart.js
- Google charts
- nvd.js - based on d3.js
- plotly. - Javascript, Python libraries
  - syntax similar between 2 languages.

Flask HTTP
- use in Backend - route files to render website from server to computer via
- written in Python — scikit-learn, Pandas, numPy

e.g.
worldbank.py - from worldbankapp import app
app.run(host='0.0.0.0', port=3001, debug=True)

worldbankapp/__init__.py - from flask import Flask
app = Flask(__name__)

from worldbankapp import routes

/routes.py - from worldbankapp import app
from flask import render_template
@app.route('/')
@app.route('/index')
def index():
return render_template('index.html')

@app.route('/project-one')
def project-one():
return render_template('project-one.html')

/template/index.html, project-one.html

Pass information back + forth from backend to frontend.
render_template('index.html', data_set = data)

index.html:
allows with {{ data_set }}

Flask comes with template engine - Jinja.
{% for tuple in data_set %}
 {{ tuple }}
{% end_for %}

With Plotly: routes.py
import plotly.graph_objs as go
import plotly, json

graph_one = [go.Scatter(
x = data[0][1],
y = data[0][2],
mode = 'lines',
name = country
)]

layout_one = dict(title = ' ',
xaxis = dict(title = ' ',
autotick = False,
tick0 = 1995,
dtick = 25),
yaxis = dict(title = ' '))

figures = []
figures.append(dict(data = graph_one,
layout = layout_one))

```
ids = ['figure-{}'.format(i) for i, _ in enumerate(figures)]

figuresJSON = json.dumps(figures, cls=plotly.utils.PlotlyJSON
                                              Encoder)

@app.route('/')
@app.route('/index')
def index():
    return render_template('index.html', ids=ids,
                           figuresJSON=figuresJSON)
```

index.html:

```
<script type="text/javascript">
    var figures = {{figuresJSON | safe}};
    var ids = {{ids | safe}};
    for (var i in figures) {
        Plotly.plot(ids[i],
            figures[i].data,
            figures[i].layout || {});
    }
</script>
```

Connect Flask to SQL database.

Heroku - PaaS
       - free tier

```
web-app/ worldbank.py
       / worldbankapp
       / scripts
       / data
```

```
python3 -m venv worldbankenv      -create virtual env.
source worldbankenv/bin/activate
```

pip install flask pandas plotly gunicorn

cd web-app
curl https://heroku/.... - install Heroku
heroku login

worldbank.py - delete line app.run()
touch Procfile
web gunicorn worldbank:app

pip freeze > requirements.txt
git init
git add .
git commit -m 'commit'

git config user.email email
git config user.name 'name'

heroku create name-of-webapp --buildpack heroku/python
git remote -v
git push heroku master
→ https://name-of-webapp.herokuapp.com

set in environmental variable
heroku config:set SLUGIFY_USES_TEXT_UNIDECODE=yes
heroku config:set AIRFLOW_GPL_UNIDECODE=yes
heroku config

4. Data Engineering
Lesson 2: ETL Pipelines

ETL - extract, transform, load.
ELT - extract, load, transform

→ Extract:
1. CSV
2. JSON
3. XML
4. SQL
5. text files

1. CSV:
```
pd.read_csv('    ', dtype=str, skiprows=4)
```

2. JSON:
```
pd.read_json('    ', orient='records')
```

or, import json
```
with open('    ') as f:
    json_data = json.load(f)
```

3. XML:
```
from bs4 import BeautifulSoup
with open('    ') as f:
    data = BeautifulSoup(f, "lxml")

for record in data.find_all('record'):
    for record in record.find_all('field'):
        fields_dict[record['name']].append(record.text)
```

4. SQL
```
conn = sqlite3.connect('    ')
pd.read_sql('  query  ', conn)
```

or, engine = create_engine('sqlite://path.db')
```
pd.read_sql('  query  ', engine)
```

API:

```
import requests
import pandas as pd

url = 'http://...'
r = requests.get(url)
r.json()
pd.DataFrame(r.json()[1])
```

→ Transform:

- Combining data

```
pd.concat([df1, df2, df3])

pd.melt(df1,
        id_vars=[ ],
        var_name=' ',
        value_name=' ')

df.merge(df2, on=[' ', ' '], how=' ')
```

- Cleaning Data
  - missing values
  - inconsistencies
  - duplicate data
  - incorrect encoding

```
df[[' ', ' ']].drop_duplicates()
              .unique()
df[' '].str.replace(' ', ' ')
pd.numeric(df[' '])

pd.to_datetime('January 1st, 2017')
               ('01/01/2017 5:30', format='%d/%m/%Y %H:%M')
```

- encoding:

```
import chardet
with open('mystery.csv', 'rb') as f:
    print(chardet.detect(f.read()))
```

- missing data:
  - for ML, can't have missing values.
    but decision trees, can.

  - impute missing data
    - mean substitution, median, mode
    - forward fill, backward fill - best if ordered by time.

```
df.transform(lambda x: x.fillna(x.mean()))

df.sort_values('year').fillna(method='ffill')
```

    - ordered fill forward/backward will have different results.

- duplicate data:
  - check for similar names, abbreviation, ...

- dummy variables

```
df.replace('  ', ' ')
df.replace('^(\' \)', '', regex=True)
```

- outliers
  - PCA
  - clusters
  - quantile = df.quantile(0.25) = Q1
    Q3 = df.quantile(0.75)
    Interquartile range, IQR = Q3 - Q1
    max_value = Q3 + 1.5 IQR
    min_value = Q1 - 1.5 IQR
    - df[(df < max_value) & (df > min_value)]

Tukey_rule

find extremes

2D analysis

```
model = LinearRegression()
model.fit(df.values.reshape[-1, 1],
          df.values.reshape[-1, 1])

inputs = np.linspace(1, 2*10^6, num=50)
predicts = model.predict(inputs.reshape(-1, 1))

df.plot(x, y, kind='scatter')
plt.plot(inputs, predicts)
```

- Feature scaling
  - Rescaling = scale range from 0 to 1
    - keep distribution.
  - Standardize - mean = 0, standard deviation = 1
    - also keep distribution.
  - $x = \frac{x - x_{min}}{x_{max} - x_{min}}$

- Feature Engineering
  - manipulate existing features to create new features
    - mean, max, min
    - power, add, (subtract).

→ Load!

```
df.to_json('    ', orient='records')
df.to_csv('    ', index=False)
df.to_sql(tablename, con=conn, if_exists='replace')

pd.read_sql('SELECT * FROM tablename', con=conn)

conn.commit()
conn.close()
```

```
conn = sqlite3.connect('    .db')
cur = conn.cursor()
cur.execute(sql_query)
conn.commit()

cur.execute("SELECT * FROM table")
cur.fetchall()

conn.close()

cur.execute("DROP TABLE IF EXISTS test")
cur.execute("CREATE TABLE test
            (colA, colB, colC REAL, PRIMARY KEY
             TEXT, INTEGER,      (colA, colB));")

for index, values in df.iterrows():
    colA, colB, colC = values
    query = 'INSERT INTO test (colA, colB, colC)
             VALUES ("{}", {}, {});'.format(colA, colB, colC)
```

ETL process
1. read csv, create tables
2. each row → dataframe → processing → row
3. row → insert into sql table

Lesson 3: NLP Pipelines

NLP Pipelines

1. Text processing
   - cleaning
   - normalize
   - tokenize
   - stop word removal
   - part of speech tagging
   - named entity recognition
   - stemming & lemmatization

2. Feature extraction - extract + produce features
   - bag of words
   - TF-IDF
   - word embeddings

3. Modeling - design a statistical or ML model

1. Text Processing
   - text from web, PDFs, word, book scans
   - remove punctuation, common words (and, a, the, of,)

- Cleaning:

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(r.text, 'html5lib')
```

summaries = soup.find_all("div", class_="course-summary-card")

{ import requests
  r = requests.get('url')

summaries[0].select_one("h3 a").get_text().strip()

summaries[0].select_one("div [data-course-short-summary]").get_text()
.strip()

- Normalization:
  text = text.lower() - convert to lowercase

```
import re
text = re.sub(r"[^a-zA-Z0-9]", " ", text)
```
- remove punctuation

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
```

- Tokenization:
  - splitting each sentence into individual words.

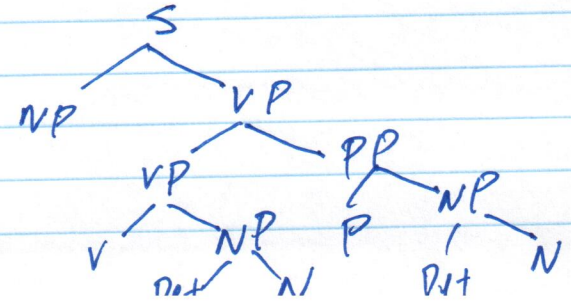
```
words = text.split()

from nltk.tokenize import word_tokenize, sent_tokenize
words = word_tokenize(text)
sentences = sent_tokenize(text)
```

- Stop Word Removal:

```
from nltk.corpus import stopwords
print(stopwords.words("english"))

                                corpus = collection of text

words = [w for w in words if w not in stopwords.words(
                                "english")]
```

- Part of Speech Tagging

```
from nltk import pos_tag
sentence = word_tokenize("I always lie down to
                          tell a lie.")
pos_tag(sentence)
  - noun, verb, pronoun ...

my_grammar = nltk.CFG.fromstring("""  """)
parser = nltk.ChartParser(my_grammar)
sentence = word_tokenize("  ")
for tree in parser.parse(sentence):
    tree.draw()
```



• Named Entity Recognition

```
from nltk import pos_tag, ne_chunk
from nltk.tokenize import word_tokenize

ne_chunk(pos_tag(word_tokenize("          ")))
```

S

PERSON — ORGANIZATION — GPE (Geopolitical Entity)

• Stemming:

```
from nltk.stem.porter import PorterStemmer
stemmed = [PorterStemmer().stem(w) for w in words]
```

remove - 'ing
- 'ed
- 'es

final word may be not a word

• Lemmatization:

is
being
was

} is

final word is a word based on dictionary.

```
from nltk.stem.wordnet import WordNetLemmatizer
lemmed = [WordNetLemmatizer().lemmatize(w)
          for w in words]
```

, pos='v' → verbs

2. Feature Extraction

• Bag of Words

unordered collection or set

~~document~~ sentence	word 1	word 2	3	4	5 ...	
1	1	1	0	0	1 ...	Document-Term Matrix
2	n	n				
3						
4						

↳ term frequency

dot product - more similar 2 documents greater number
Can also true for non-matching terms
$a \cdot b = \sum_i a_i b_i$

cosine similarity - $\cos(\theta) = \frac{a \cdot b}{\|a\| \|b\|}$

$-1 < \cos\theta < 1$

most dissimilar ↘ most similar

• TF-IDF
- some words have more weights or importance.

$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$

term frequency: $\frac{count(t,d)}{|d|}$

inverse document frequency: $\log\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right)$

- normalize or smooth values

Counter Vectorizer (Bag of Words)

```
from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer(tokenizer = tokenize)
x = vect.fit_transform(corpus)
x.toarray()
vect.vocabulary_
```

```
from sklearn.feature_extraction.text import TfidfTransformer
transformer = TfidfTransformer(smooth_idf = False)
tfidf = transformer.fit_transform(X)
tfidf.toarray()
```

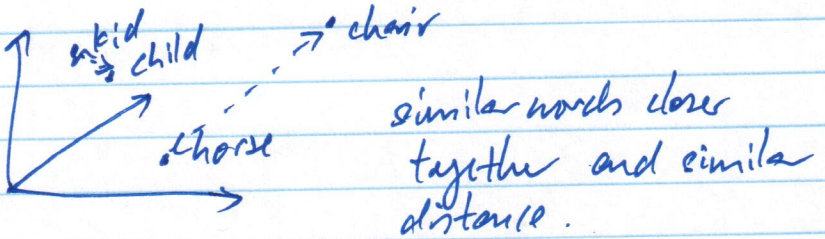
TfidfVectorizer = CountVectorizer + TfidfTransformer

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
x = vectorizer.fit_transform(corpus)
x.toarray()
```

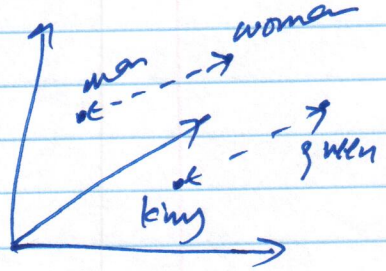
- One-Hot Encoding
  - treat each word like a class
  - create vector, can get large

	word 1	2	3	4
doc 1	1	0	0	0
2	0	1	1	0
3				

- Word Embedding
  - limit to fix size vector



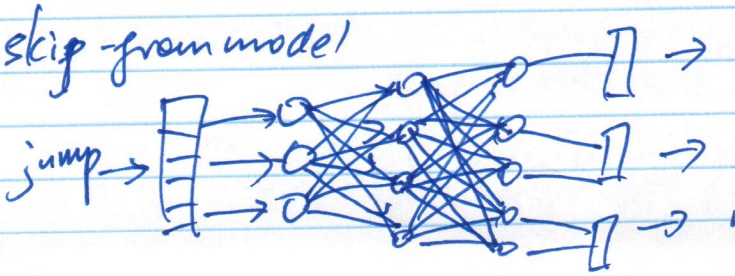
similar words closer together and similar distance.



3. Modeling
   - fit parameters to training data
   - use optimization prodcure
   - make predictions on unseen data

- Word2Vec
  - predict neighboring words given a word (continuous bag of words)
  - predict given word using neighboring words (continuous skip-gram)

skip-gram model



- Glove - Global Vectors for Word Representation:

$w_i \cdot w_j = P(j|i)$ ← co-occurance probability

(w_i: context word, w_j: target word)

- Embeddings for Deep Learning:
  - words that occur in same context have similar meaning
    = Distributional Hypothesis

- t-SNE
  - t-distributed stochastic neighbor embedding
  - maintains relative distance between objects
    (similar words have similar distances.
    explore grouping of similar words / images.

Lesson 4: Machine Learning Pipelines

e.g.
```
def tokenize(text):
    detected_urls = re.findall(url_regex, text)
    for url in detected_urls:
        text = text.replace(url, "placeholder")
    tokens = word_tokenizer(text)
    lemmatizer = WordNetLemmatizer()

    clean_tokens = []
    for tok in tokens:
        clean_tok = lemmatizer.lemmatize(tok).lower()
        clean_tokens.append(clean_tok).strip()

    return clean_tokens
```

Pipelines:

```
pipeline = Pipeline([
            ('vect', CountVectorizer()),          } transform
            ('tfidf', TfidfTransformer()),
            ('clf', RandomForestClassifier()),    } classifier
           ])

pipeline.fit(X_train, y_train)

predicted = pipeline.predict(X_test)      } predict last if last is classifier.
```

Estimator – any object that learns from data.
- e.g. classification, regression, clustering, transformer.
- must have fit method

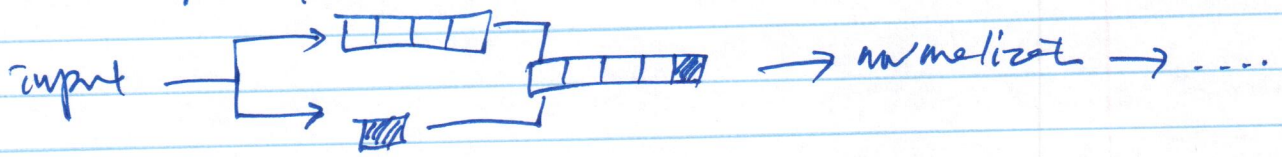
Transformer – transform method to apply transformation model to new data.
- can include cleaning, reducing, expanding, or generating features.

Predictor – predict method to predict test data.
has fit method to train model on training data.

GridSearch – automates process of testing different hyper parameters to optimize model.
- tune our entire pipeline

Prevent data leakage – contains all steps within each fold of cross-validation

Feature Union:
- perform steps in parallel and take union of results for next step.




```
pipeline = Pipeline([
    ('features', FeatureUnion([
        ('nlp-pipeline', Pipeline([
            ('vect', CountVectorizer()),
            ('tfidf', TfidfTransformer())
        ])),
        ('text_len', TextLength Extractor())
    ])),
    ('clf', RandomForestClassifier())
])

pipeline.fit(Xtrain, y_train)
predicted = pipeline.predict(Xtest)
```

custom transformer

Custom Transformer:

```
from sklearn.base import BaseEstimator, TransformerMixin
class TenMultiplier(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return X * 10
```

- Fit method - X = 2d array for feature data
  y = 1d array for target
  - transformer method return self

- Transform method - does the transformation using X.

```
multiplier = TenMultiplier()
X = np.array([ , , , ])
multiplier.transform(X)
```

```
class CaseNormalizer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return pd.Series(X).apply(lambda x: x.lower()).values

    def __init__(self):
        pass

case_normalizer = CaseNormalizer()
X = np.array([ , , , ])
case_normalizer.transform(X)
```

```
FunctionTransformer:
    from sklearn.preprocessing import FunctionTransformer
    transformer = FunctionTransformer(np.log1p, validate=True)
    X = np.array([ , , , ])
    transformer.transform(X)
```

GridSearch:
- using pipeline, run search on entire ~~pp~~ workflow
- define grid of parameters, scores each combination with cross-validation, determine parameter that results in optimal model.
- also works on 1 classifier.

```
parameters = {
    'kernel': ['linear', 'rbf'],
    'c': [1, 10]
}

svc = SVC()
clf = GridSearchCV(svc, parameters)
clf.fit(X_train, y_train)
```

- avoid data leakage using pipeline,

```
pipeline = Pipeline([
                ('scaler', StandardScaler()),
                ('clf', SVC()
            ])

parameters = {
                'scaler__with_mean': [True, False],
                'clf_kernel': ['linear', 'rbf'],
                'clf__c': [1, 10]
            }
```

standardization happens when grid search starts. For each cross-validation, scaling is done only on data that is split and to be trained on, preventing leakage.

```
def main():
    X, y = load_data()
    X_train, X_test, y_train, y_test = train_test_split(X, y)

    model = build_model()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    display_results(model, y_test, y_pred)


def build_model():
    pipeline = Pipeline([
            ('features', FeatureUnion([
                (                                   ),
                {       ,                           },
                {       ,
            ])),
            (                                       ),
        ])),
        (                                           )
    ])

    parameters = {
            ]

    cv = GridSearchCV(pipeline, param_grid=parameters)

    return cv

def display_results(cv, y_test, y_pred):
    labels = np.unique(y_pred)
    confusion_mat = confusion_matrix(y_test, y_pred, labels)
    accuracy = (y_pred == y_test).mean()
```

5. Experiment Design + Recommendation

Lesson 2: Experimental Design concepts

Correlation does not imply Causation

1. what is experiment?
2. what are types of experiments?
3. How to measure outcome?
4. common pitfalls of experiment design?

Key feature of experiment
1. Comparison between groups
2. Control for other variables.
   - randomization

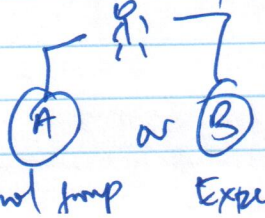
1. Types of Experiment.
   Control on variables

None → Some → High

- None: observational study
  - lack control due to ethical concerns
  - directionality of cause might be uncertain
- Some: quasi-experiment
- High: experiment
  - compare outcomes between ≥ 2 groups.

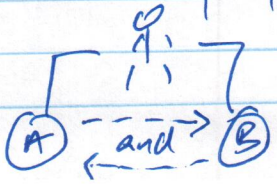
A/B test - compare performance between group A and B.

Between-subjects design:
1 condition per participant.



Control group Experiment group

Within-subject design:
many conditions per participant



Factorial designs.
- control, X, Y, X and Y

2. Types of sampling
- Simple random sampling
  - all populat. have equal chance of selection
- Stratified random sampling
  - divide populat. into distinct subgroups before sampling.
  - proportional sampling - maintain distribut
  - non proportional sampling - fix number sample from each group.
- probabilistic sampling - sample representative of whole populat.

Convenience Sample - limited to group of people
  - e.g. college students, elderly, military.

3. Measure outcomes
- objective features to evaluate performance - evaluat. metrics
- goal of study must be separate from evaluat. metrics
  - metric is not main point of study
  - ~~metric~~ implicat. of metric relative to goal
  - why want to run experiment.

- metrics - key results (KR)
  - key performance indicators (KPI)
  - operational definition
- objectives - constructs

User Funnel - path/steps taken by user from start of experience to desired distinat. state.
- drop off in users as move from step to step. Motivate experiments to improve drop offs.

Unit of diversion - observation unit by which users are split into experimental groups.

1. Event-based diversion
   - experiment condition randomly rolled
   - ignores previous visits

2. Cookie-based diversion
   - stored on user device
   - determines experimental condition
   - subverted by deleting cookies / anonymous browsing.

3. Account-based diversion
   - randomly divided user IDs into condition
   - requires accounts + logged in.
   - limit pool of data, risk personal identifiable information.

Consider consistency of experience when selecting diversions.

Evaluation Metrics
- Features that provide an objective measure of success of an experimental manipulation.
- e.g. proportion of visitors made purchase.
  average amount purchased.

Invariant metrics
- Features that objectively check for equivalence between experiment groups, makes sure experiment is running as expected, no biases.
- e.g. number of cookies, logged in users.

4. Controlling Variables
- Need to have 1. one variable feature of interest.
  2. none other features changes.

all
- if unable to control other features, then susceptible to confounding variables.
  - outcome observed may be due to 3rd variable, rather than feature of interest.
  - causal relationship between 3rd variable/feature of interest, ~~or~~ indirect relationship mediated by 3rd, intermediate variable

- direction of effect - A to B
  B to A
  A, B to C

5. Experiment validity - degree where experiment actually accomplishes conclusion of study.

1. Construct Validity
   - objective metric results support goals of study.

2. Internal Validity
   - claims of causality can be supported.

3. External Validity
   - results can be generalized to cases outside experiment.

6. Bias
   - systematic errors that affect interpretation of results.

- Sampling bias - not representative of population.
  - e.g. self-selection bias, survivor bias.
- Novelty bias - change in participant behavior solely due to novel stimuli.

- order biases – condition order whether occur first (primacy bias) or last (recency bias).
  - avoid by randomizing order of conditions.

- experimenter bias – presence of experimenter introduces bias from subject, or from experimenter him/herself.
  - blinding participants and experimenters from condition to help avoid subconscious bias.

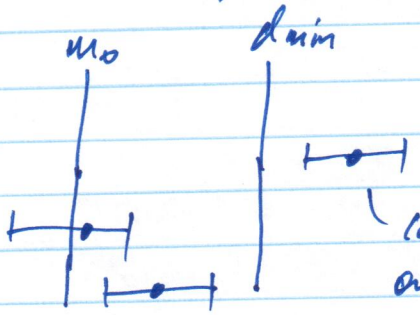
7. Ethics
   - minimize participant risk – physical, emotional.
   - clear benefits for risks taken.
   - provide informed consent
     - if deception is needed, must debrief at end of experiment.
     - included in terms of agreement.
   - handle sensitive data appropriately.
     - personal identifiers must be anonymized.
     - survey + census results aggregated to avoid tracing outcomes back to individual person.

   - Institutional Review Board (IRB) to approve proposals.

S – specific – goals are specific.
M – measureable – outcomes measurable
A – achievable – steps + goals realistic
R – relevant – purpose
T – timely – reasonable time frame

Lesson 3: Statistical Considerations in Testing

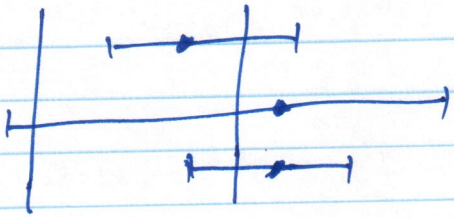
Practical significance – level of effect need to observe for experiment to be a true success.



μ_0: null statistical value
d_{min}: practical significance bound.

confidence interval outside d_{min}, experiment successful.

within d_{min}, experiment failed



- uncertainty as confidence interval straddles d_{min} bounds.
- option to collect more data

Statistical power – given true mean, probability of rejecting null hypothesis.

Hypothesis Testing

null hypothesis : $H_0: \mu = 191$
research hypothesis : $H_1: \mu > 191$

$n = 100$
$\bar{X} = 197.1$
stdev $= 25.6$

} does new sample data support null or research hypothesis?

probability using Central Limit Theorem

$$P(\bar{X} > 197.1) = P\left(z > \frac{\bar{X} - \mu_0}{s/\sqrt{n}}\right) = P(z > 2.38)$$
$$= 1 - 0.9913$$
$$= 0.0087$$

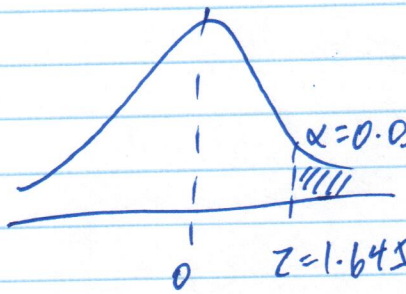
low chance of null hypothesis.

$n = 100$
$\bar{x} = 192.1$
$sd = 25.6$

$$P(\bar{X} > 192.1) = P\left(z > \frac{\bar{X} - \mu_\phi}{\frac{sd}{\sqrt{n}}}\right) = P(z > 0.43)$$
$$= 1 - 0.6614$$
$$= 0.3336$$

33.36% probability ∧ sample means > 192.1.

α - level of significance = P(Type I error)
= P(Reject H_ϕ | H_ϕ is true)



for $\alpha = 5\%$, $z > 1.645$
reject H_ϕ

Step 1: Test hypothesis.
1. Upper-tailed test
   - $H_1 : \mu > \mu_\phi$

2. Lower-tailed test
   - $H_1 : \mu < \mu_\phi$

3. 2-tailed test
   - $H_1 : \mu \neq \mu_\phi$

Step 2: Test statistic

$$z = \frac{\bar{x} - \mu_\phi}{\frac{s}{\sqrt{n}}}$$

Step 3: Set up Decision Rule.
1. type of test
   upper - reject H_ϕ if $z \geq z_\alpha$
   lower - " " " $z \leq z_\alpha$
   2-tailed - " " " $z_\alpha \leq z \leq z_\alpha$

2. type of test statistics
   - z standard norm distribution
   - t distribute.

3. α - level of significance.

Step 4: Compute test statistics

$H_\phi : p_1 - p_2 = \phi$

$$z = \frac{(\hat{p}_1 - \hat{p}_2)}{\sqrt{\hat{p}(1-\hat{p})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

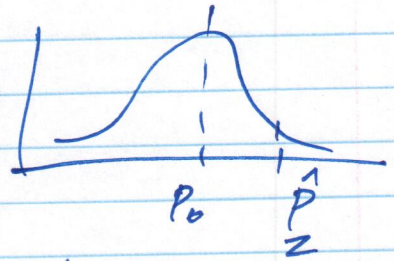
where $\hat{p} = \frac{Y_1 + Y_2}{n_1 + n_2}$

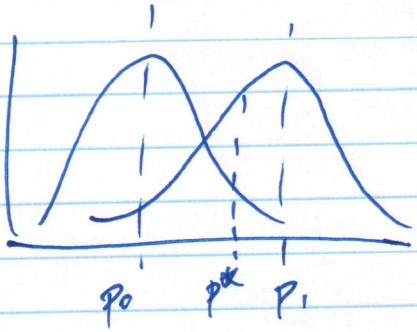
sample proportion

standard deviation $SE_0 = \sqrt{\frac{p_0(1-p_0)}{n}}$

$\hat{p} = \frac{Y}{n}$

$$z = \frac{\hat{p} - p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}} = \frac{\hat{p} - p_0}{SE_0}$$





Number of samples

$p^* - p_\phi = z_{1-\alpha} SE_\phi$ — (1)

$p_1 - p^* = -z_\beta SE_1$ — (2)

$p_1 - p_\phi = z_{1-\alpha} SE_\phi - z_\beta SE_1$

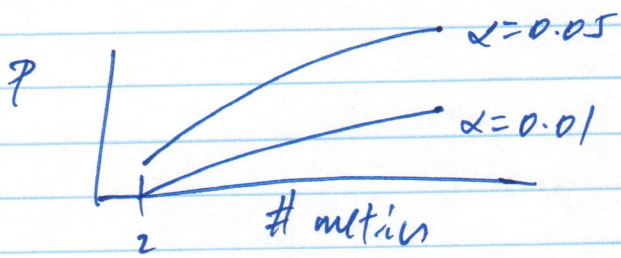
$$p_1 - p_\phi = \frac{z_{1-\alpha}\sqrt{p_0(1-p_0)} - z_\beta\sqrt{p_1(1-p_1)}}{\sqrt{n}}$$

$$n = \left(\frac{z_{1-\alpha}\sqrt{p_0(1-p_0)} - z_\beta\sqrt{p_1(1-p_1)}}{p_1 - p_\phi}\right)^2$$

Dummy test - experiment run against equivalent groups
- A-A test
- provide distribution data if historical data not available.
- test for any errors
- requires time + resources.

- quantile_ci
- rank-sum test (mann-whitney)
  - `scipy.stats.mannwhitneyu(x, y, alternative='greater')`
- sign test

- Probability of making at least one Type I error
  $\Rightarrow \alpha_{over} = 1 - (1 - \alpha_{ind})^n$



introduce correction factor, $\alpha_{ind} = \frac{\alpha_{over}}{n}$
Bonferroni correction

Sidak correction, $\alpha_{ind} = 1 - (1 - \alpha_{over})^{1/n}$

$p = 0.5$

$stderr = \sqrt{p * (1-p) * \left(\frac{1}{n_1} + \frac{1}{n_2}\right)}$

$z = \frac{(n_1 + 0.5) - (p * n)}{stderr}$ or $\frac{p_1 - p_2}{SE}$

$p = 1 - stats.norm.cdf(z)$

Lesson 4: A/B Testing Case Study

Goal: Revising structure of homepage will increase number of people download software, eventually number of people that purchase a license.

Funnel:
1. homepage
2. Electron with additional information gallery
3. page for users to download software
4. " " " purchase license
5. support page with documentation + FAQ for software.

- user account required.
- program free to use for 7 days trial.
- end of 7 days, direct user to a dialog box to license page.
- purchase code, receive unique code to register account with program.

A/B testing – Control = group A
– experiment = group B

Unit of diversion
- event-based – good for when changes are not user visible.
- account-based – requires user accounts created.
- cookie-based – assign on first page view.
  track whether - homepage is reached or not.
  - account is reached or not.

Metrics – number of cookies on – homepage
– download page
– account creation page
number of purchases – user account database.

invariant metric - number cookies @ homepage
evaluation metric - ratio # downloads / # cookies
- ratio # licenses / # cookies

Lesson 6: Introduction to Recommendation Engines

Recommendation types:
1. Knowledge-based
2. Collaborative filtering-based — model-based collaborative filtering
3. Content-based — neighborhood based

Similarity metrics:
1. Pearson's correlation coefficient
2. Spearman's " "
3. Kendall's Tau
4. Euclidean Distance
5. Manhattan Distance

Business cases for recommendations:
1. Relevance
2. Novelty
3. Serendipity
4. Increased diversity

Rank-based recommendation
- orders based on metric - cost, highest ratings, most popular

- Knowledged-based
  - users provide preferences & filter to make recommendation.
- Collaborative filtering - recommendations based on interaction between users + items.

- Content-based – use information about users w items to assist in recommendations.

Collaborative Filtering:
1. model-based
   - ML modeling

2. Neighborhood-based
   - Similarity-Based methods
     - Pearson's correlation
     - Spearman's correlation
     - Kendall's tau

   } range -1 to $+1$
   (-1: most dissimilar, $+1$: most similar)

   - Distance Based methods
     - Euclidean distance
     - Manhattan distance

   } range 0 to ∞
   (0: most similar, ∞: most dissimilar)

Pearson's, $CORR(x,y) = \dfrac{\sum_i^N (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i^N (x_i - \bar{x})^2 (y_i - \bar{y})^2}}$

Spearman's, $SCORR(x,y) = \dfrac{\sum_i^N (x_i^r - \bar{x}^r)(y_i^r - \bar{y}^r)}{\sqrt{\sum_i^N (x_i^r - \bar{x}^r)^2 (y_i^r - \bar{y}^r)^2}}$

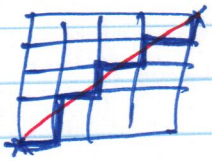
$x \to x^r$; ranked
$y \to y^r$; ranked

Kendall's Tau, $tau = \dfrac{2}{n(n-1)} \sum_{i<j} sign(x_i^r - x_j^r)\, sign(y_i^r - y_j^r)$

$$sign(x_i^r - x_j^r) = \begin{cases} -1, & x_i^r < x_j^r \\ 0, & x_i^r = x_j^r \\ 1, & x_i^r > x_j^r \end{cases}$$

eucidian distance, $EUC(x,y) = \sqrt{\sum_i^N (x_i - y_i)^2}$

manhatten distance, $manhat(x,y) = \sum_i^N |x_i - y_i|$



euclidian

Movie recommendations example:

1. Data cleaning
2. Rank by mean ratings
   count ratings
   data ratings
3. Filter by year
   genre

} knowledge-based

1. user-item matrix

	movieid 1	2	3
user 1	rating	"	"
2	"	"	"
3	"	"	"

} user_by_movie

• movies_watched (user_id)

• movies_seen } dictionary of each user (key), list of movies (value).

• movies_to_analyze } dictionary of users to >2 movies.

• Compute_euclidean distance between user 1, 2

movies 1 = movies_to_analyze [user 1]

movies 2 = " " [user 2]

same_movies = np.intersect1d (movies 1, movies 2, assume_unique = True)

```
df = user_by_movie.loc[[user1, user2], same_movies]
dist = np.linalg.norm(df.loc[user1] - df.loc[user2])
```

- find_closest_neighbors(user)
  - ranked by euclidean distance,

- movies_liked(user_id, min_rating=7)

```
user_items[(user_items['user_id'] == user_id) &
           (user_items['rating'] >= min_rating)]
          ['movie_id']
```

- movies_names(movie_ids)

```
make_recommendation(user, num_rec=10)
    movies_seen = movies_watched(user)
    closest_neighbors = find_closest_neighbors(user)

    recom = np.array([])
    for neighbor in closest_neighbors:
        neighbor_liked = movies_liked(neighbor)
        new_recom = np.setdiff1d(neighbor_liked,
                                 movies_seen,
                                 assume_unique=True)
        recomm = np.unique(np.concatenate([
                           recomm, new_recom],
                           axis=0))

    if len(recomm) > num_rec-1:
        break

    movie_names(recomm)
```

Content-based recommendation:
matrix of similar movies by dot product
(range from 0 to ∞)
↑ not similar
↓ more similar

movies [ElectMovies, ..., genre, ...]
dot_product = movies.dot(np.transpose(movie))

↓

movie_	1	2	3	...
movie_1	0	1	1	
2	1	2	0	
3	1	7		...

movies + reviews user has seen

```
for user in users:
    reviews_temp = ranked_reviews[ranked_reviews['user_id'] ==
                                  user]
    movies_id_temp = np.array(reviews_temp['movie_id'])
    movies_temp = np.array(get_movie_names(movies_id_temp))
```

↳ `movie_list = list(movies[movies['movie_id'].isin(movie_id)]['movie'])`

```
recs = defaultdict(set)
for movie_id in movies_id_temp:
    rec_movies = find_similar_movies(movie_id)
    temp_recs = np.setdiff1d(rec_movies, movie_temp)
    recs[user].update(temp_recs)
```

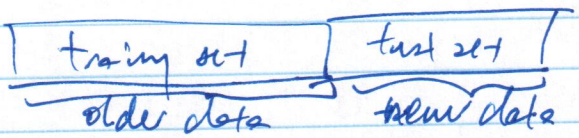
```
movie_idx = np.where(movies['movie_id']
                     == movie_id)[0][0]
similar_idx = np.where(dot_product[movie_idx]
                       == np.max(dot_product[movie_idx]))[0]
similar_movies = np.array(movies.iloc[similar_idx,]['movie'])
```

Ratings
- interactions - what questions to ask?
  - how many to ask?
  - what type of scale to use?
    - 0 to 1 } binary. no mid-way/neutral
    - 1 to 4 }
    - 1 to 7 } neutral ratings.
    - 1 to 3 }

Goals of recommendation system
1. Relevance - relevant to users
2. Novelty - something user is not already exposed.
3. Serendipity - something user unknown to, different, but likes
4. Diversity - diverse things to recommend, rather than a bunch of similar ones.

Lesson 7: Matrix Factorization for Recommendation:

Test recommendation engine before going live. - offline testing.

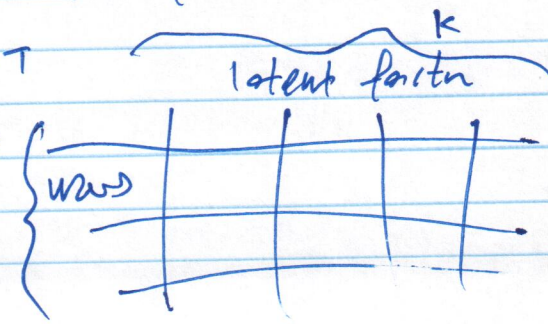


compare predicted rating to actual.

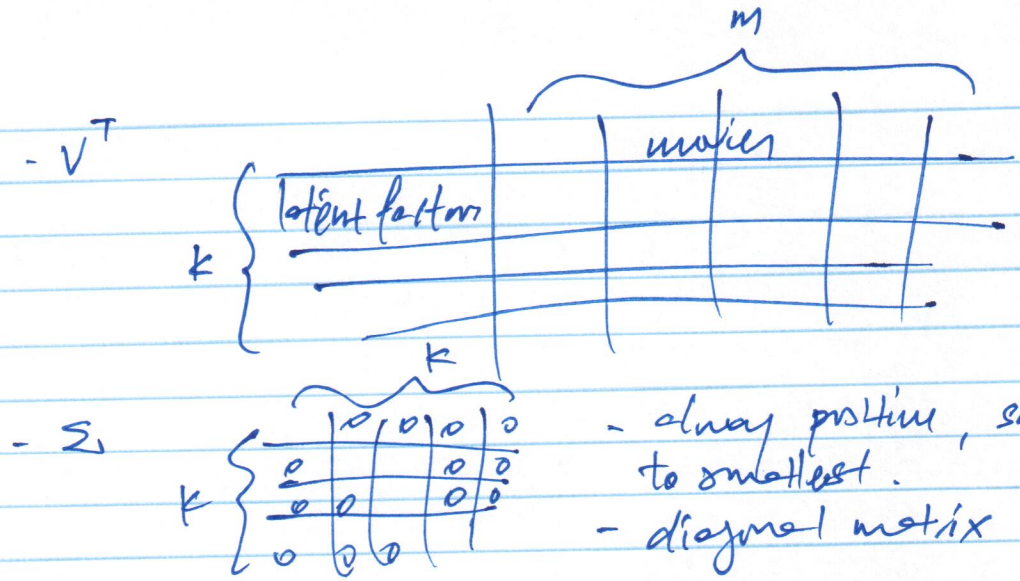
Latent factors - not observed in data, but inferred based on relationships that occur.
- use singular value decomposition (SVD) to find latent factors common to movies + customers.

SVD - $U \Sigma V^T$

- $U \rightarrow$



user-latent factors matrix



- V^T

- Σ

- always positive, sorted largest to smallest.
- diagonal matrix

- $U \Sigma V^T$ = user-item (movie) matrix

u, s, vt = np.linalg.svd(user_movie)

(will not work if any NaN present.

total variability of data = $\sum s^2$

variability of 1st n components = $\sum_i^n s_i^2$

% variability of 1st n = $\frac{\sum_i^n s_i^2}{\sum_i^N s_i^2}$

→ reduce # of latent factors by their % variability

predicted = $u \cdot s \cdot v^T$

squared error = $\sum \sum (\text{predicted} - \text{actual})^2$

latent factors - one not labeled
- only know their weights

1. Solve SVD analytically treating it as a closed loop solution. But only works without missing data.

2. Solve SVD using gradient descent method to handle missing data
   - FunkSVD $UV^T = A$

- Gradient Descent

User matrix: $\frac{\partial}{\partial u_i}(y - uv)^2 = 2(y - uv)v_i$

movie matrix: $\frac{\partial}{\partial v_i}(y - uv)^2 = 2(y - uv)u_i$

update: $u^i_{new} = u^i_{current} + 2\alpha(y - uv)\,v^i_{current}$

- Only apply to non-NaN values in U and V

```
def FunkSVD(ratings_mat, latent_features=4, learning_rate=0.0001,
                                                iters=100):
    n_users = ratings_mat.shape[0]
    n_movies = "  "  "  [1]
    num_ratings = np.count_nonzero(~np.isnan(ratings_mat))

    user_mat = np.random.rand(n_users, latent_features)
    movie_mat = "  "  "  (latent_features, n_movies)

    sse_accum = 0

    for iter in iters:                    sse_accum = 0
        for i in range(n_users):
            for j in range(n_movies):
                if ratings_mat[i,j] > 0:
                    diff = ratings_mat[i,j] -
                           np.dot(user_mat[i,:], movie_mat[:,j])

                    sse_accum += diff**2

                    for k in range(latent_features):
                        user_mat[i,k] += learning_rate * 2 * diff
                                         * movie_mat[k,j]

                        movie_mat[k,j] += learning_rate * 2 * diff
                                          * user_mat[i,k]
```

```
def main():
    import recommender_functions as rf
    import recommender as r
    rec = r.Recommender()
    rec.fit(reviews_path = "train_data.csv",
            movies_path = "movies_clean.csv",
            learning_rate = 0.01
            iters = 1)

    rec.predict_rating(user_id = 8,
                       movie_id = 2844)

    rec.make_recommendation(8, 'user')
    "    "    "    (1, 'user')
    "    "    "    (1853728)
    "    "    "    (1)


class Recommender():
    def __init__(self):

    def fit(self, reviews_path, movies_path, learning_rate,
            latent_features = 12, iters = 100):

        self.reviews = pd.read_csv(reviews_path)
        self.movies = pd.read_csv(movies_path)

        user_item = self.reviews[['user_id', 'movie_id',
                                  'rating', 'timestamp']]
        self.user_item_df = user_item.groupby(['user_id',
                                               'movie_id'])['rating']
                                    .max().unstack()
        self.user_item_mat = np.array(self.user_item_df)

        self.latent_features = latent_features
        self.learning_rate = learning_rate
        self.iters = iters
```

```
self.n_users = self.user_item_mat.shape[0]
self.n_movies = "      "      "      [1]

self.num_ratings = np.count_nonzero(~np.isnan(self.user_item_mat))
self.user_ids_series = np.array(self.user_item_df.index)
self.movie_ids_series = np.array(self.user_item_df.columns)

user_mat = np.random.rand(self.n_users, self.latent_features)
movie_mat = np.random.rand(self.latent_features, self.n_movies)

sse_accum = 0

for iteration in range(self.iters):
    sse_accum = 0

    for i in range(self.n_users):
        for j in range(self.n_movies):
            if self.user_item_mat[i,j] > 0:

                diff = self.user_item_mat[i,j] -
                       np.dot(user_mat[i,:], movie_mat[:,j])

                sse_accum += diff**2

                for k in range(self.latent_features):
                    user_mat[i,k] += self.learning_rate * 2 * diff *
                                     movie_mat[k,j]

                    movie_mat[k,j] += self.learning_rate * 2 * diff *
                                      user_mat[i,k]

    print("%d \t\t %f" %((iteration+1), sse_accum/self.num_ratings))
```

FunkSVD

```
        self.user_mat = user_mat
        self.movie_mat = movie_mat

        self.ranked_movies = rf.create_ranked_df(self.movies,
                                                 self.reviews')


    def predict_rating(self, user_id, movie_id):

        try:
            user_row = np.where(self.user_ids_series == user_id)[0][0]
            movie_col = " "  ", movie_ids_series == movie_id) " "

            pred = np.dot(self.user_mat[user_row, :],
                          self.movie_mat[:, movie_col])

            movie_name = self.movies[self.movies['movie_id'] == movie_id]
                                    ['movie'].to_list()

            print("User {}, Predict rating of {}, for movie {}.".
                  format(user_id, str(pred), str(movie_name[0])))

            return pred

        except:
            print("User-movie pair not found")
            return None
```

```
def make_recommendations(self, _id, _id_type='movie',
                         rec_num=5):

    rec_ids = None
    rec_names = None

    if _id_type == 'user':
        if _id in self.user_ids_series:
            idx = np.where(self.user_ids_series == _id)[0][0]
            preds = np.dot(self.user_mat[idx,:], self.movie_mat)
            indices = preds.argsort()[-rec_num][::-1]
            rec_ids = self.movie_ids_series[indices]
            rec_names = rf.get_movie_name(rec_ids,
                                          self.movies)
        else:
            rec_names = rf.popular_recommendations(_id, rec_num,
                                                   self.ranked_movies)
            print("Offering top recommendations since user is not in db")
    elif _id_type == 'movie':
        if _id in self.movie_ids_series:
            rec_names = list(rf.find_similar_movies(_id,
                                                    self.movies))[:rec_num]
        else:
            print("Movie does not exist in db")

    return rec_ids, rec_names
```

from funkSVD

flip order

recommender_functions.py

```
def get_movie_names(movie_ids, movies_df):

    movie_list = list(movies_df[movies_df['movie_id'].isin(movie_ids)]
                              ['movie'])

    return movie_list


def create_ranked_df(movies, reviews):

    movie_ratings = reviews.groupby('movie_id')['rating']
    avg_ratings = movie_ratings.mean()
    num_ratings = movie_ratings.count()
    last_ratings = reviews.groupby('movie_id').max()['date']

    avg_ratings_df = pd.DataFrame({'avg_ratings': avg_ratings})
    num_ratings_df = "  "  ({'num_ratings': num_ratings})
    last_ratings_df = "  "  ({'last_ratings': last_ratings})

    ratings_df = pd.merge(avg_ratings_df, num_ratings_df,
                          on='movie_id', how='outer')
    ratings_df = "  "  (ratings_df, last_ratings_df,
                          on='movie_id', how='outer')

    ranked_movies = pd.merge(movies, ratings_df,
                             on='movie_id', how='right')
    ranked_movies.sort_values(['avg_ratings', 'num_ratings',
                               'last_ratings'],
                              ascending=False, inplace=True)

    ranked_movies = ranked_movies[ranked_movies['num_ratings'] > 4]

    return ranked_movies
```

find similarity based on dot product of ratings content.
highest

```
def find_similar_movies(movie_id, movies_df):

    movie_content = np.array(movie_df.iloc[:, 4:])
    dot_product = movie_content.dot(np.transpose(movie_content))

    movie_idx = np.where(movie_df['movie_df'] == movie_id)[0][0]

    similar_idxs = np.where(dot_product[movie_idx]
                            == np.max(dot_product[movie_idx]))[0]

    similar_movies = np.array(movies_df.iloc[similar_idxs]['movie'])

    return similar_movies


def popular_recommendations(user_id, n_top, ranked_movies):
    top_movies = list(ranked_movies['movie'][:n_top])
```

Recommendation Engines

1. Knowledge-based
   - filter by known parameters - year, genre ....

2. Collaborative-filtering
   - model based
   - neighbor based — similarity
     — distance
     } based on user ratings / content values

3. Content based
   - similar content - dot product of content values / ratings.
   - latent factors - SVD $A = U \Sigma V^T$
     - Funk SVD $A = UV^T$
   - recommender validation - split training, validation
   - cold start - new users do not have ratings, missing information