Summary

Intro

- Get row with index x with `df.loc[x]`. Get row number x with `df.iloc[x]`.
- Delete rows (axis=0) or columns with `df.drop('col_name', axis-1)`.
- Sort by cols with `df.sort_values(ascending=[True,False], by=['col1', 'col2'])`.
- Access month of datetime column with `df.datetime_col.dt.month`.
- Options of merging: `pd.merge(vessels, segments, left_index=False, right_on='mmsi', left_on='left_col')`
- Concatenate rows of a dataframe `pd.concat([mb1, mb2], axis=0)`
- Stacking dataframes; create one index col where the original column identifier of the data point is. `df.stack()`. The inverse is `df.unstack()s`
- Transform a table of with multiple observations to a new table showing, e.g. observations of some kind for a patient: `df.pivot(index='', columns='', values='')`
  - Similarly, with `pivot_table` we can use and aggregation funciton to create the table.
- With `pd.crosstab(df.col1, df.col2)` we can aggregate the value counts of the selected columns.
- Convert categorical data to dummy / indicator variables: `pd.get_dummies(df.col)`.
  - this also works for multiple columns at the same time

```
categorical_cols = ['animal_type', 'intake_condition', 'intake_type', 'sex_upon_intake'

num_rep = pd.get_dummies(original_data, columns=categorical_cols).drop(['outcome_type']
```

- Bucketizing data: `pd.cut(df.age, [20,40,60,80], labels=['young', ... ,'old'])`
  - `pd.qcut` divides the data by quanitles
- select rows from indices with `df.take(indices)`

Plotting

- Always add title and ax-lables with

```
plt.xlabel('bla')
plt.ylablel('bla')
plt.title('bla')
```

- Visualize two varibles: `sns.jointplot(x=df['x'], y=df['y'], kind='hex')`
  - `kind='reg'` does a scatter plot with indicated regression direction and errorbar.
  - a similar result to `kind='reg'` can be obtained with `sns.lmplot(x='col1',y='col2', data=df, hue = 'hue_col')`

- Barplots in seaborn provide errorbars by default `sns.barplot(x='labels', y='values', data=df)`
- Manual errorbars with `plt.errorbar(x, y, yerr=None)`
  - filled error area: `plt.fill_between(x, y1, y2=0)`
- Populate 4x4 subplot with histograms

```
fig, ax = plt.subplots(4,4,figsize= (8,6), sharey = True, sharex = True)

for i in range(16):
    sbplt = ax[i%4, math.floor(i/4)]
    sbplt.hist(stats_by_genre.iloc[i].values,range = [0,200],bins = 20)
    sbplt.set_title(stats_by_genre.index[i])

fig.tight_layout()

fig.text(0.4,0, "x label here")
fig.text(0,0.6, "y label here", rotation = 90)
```

- another way more for a quick subplot is

```
fig, axs = plt.subplots(1, 3, sharey=True)
data.plot(kind='scatter', x='TV', y='sales', ax=axs[0], figsize=(16, 5), grid=True)
data.plot(kind='scatter', x='radio', y='sales', ax=axs[1], grid=True)
data.plot(kind='scatter', x='newspaper', y='sales', ax=axs[2], grid=True)
```

- heatmaps with seaborn

```
df2 = pd.crosstab(df['col1'],df['col2'])
sns.heatmap(df2, annot=True)
```

- draw a loglog pdf (eg for a power law)

```
array = plt.hist(df.col,bins=1000,log=True, density=True)
plt.close()
plt.loglog(array[1][1:],array[0])
```

- `sns.pairplot(df)` is useful to describe the df as `df.describe()`

Describing data

- Test if data is exponential ('exp') or normal ('norm') distributed `statsmodels.stats.diagnostic.kstest_normal(df['col'].values, dist='norm', pvalmethod='table')`
- Personr measures linear correlation `stats.pearsonr(df['col1'],df['col2'])`. Spearmanr measures monotonic correlation, i.e. compares the rankings of the two datastreams `stats.spearmanr(df['col1'],df['col2'])`.
- Compare if two datastreams have the same mean with `stats.ttest_ind(df['col1'], df['col2'])`.

- Get p-value from a one sided binominal test `scipy.stats.binom_test(x, n=None, p=0.5, alternative='greater')` where x is the number of successful outcomes, n the number of outcomes and p the probabiliy of success.

Regression analysis

- least square model and summary printing:

```
mod = smf.ols(formula='outcome ~ C(catogory)+ continuous, data=df)
res = mod.fit()
print(res.summary())
```

- options
  - `a:b` in the formula is an interaction term
  - `a*b` is shorthand for `a+b+a:b`
  - logistic regression with `smf.logit(...)`
- postprocessing pipeline

```
variables = res.params.index
coefficients = res.params.values
p_values = res.pvalues
standard_errors = res.bse.values

#sort them all by coefficients
l1, l2, l3, l4 = zip(*sorted(zip(coefficients[1:], variables[1:], standard_errors[1:], p_val

plt.errorbar(l1, np.array(range(len(l1))), xerr= 2*np.array(l3), linewidth = 1,
             linestyle = 'none',marker = 'o',markersize= 3,
             markerfacecolor = 'black',markeredgecolor = 'black', capsize= 5)
plt.vlines(0,0, len(l1), linestyle = '--')
plt.yticks(range(len(l2)),l2);
```

- A log transform on the outcome makes the model multiplicative. Don't forget to exponentiate the coefficient again befor reasoning about the result.

Observational studies

- Default: do a propensity score matching, i.e. do the analysis on paires matched on the probability of treatment. A logistic regression can predict the propensity score for all samples.

```
mod = smf.logit(formula='treatment ~  age + educ + C(black) , data=df)
res = mod.fit()

# Extract the estimated propensity scores
df['Propensity_score'] = res.predict()
```

- Networkx can calculate a maximal matching. Hence, we have to converte the propensity scores to similarity rating $= 1 - |p_1 - p_2|$. The latter is

the edgeweigth and we add an edge between between all nodes of different groups (bipartite graph).

Supervised learning

- Use entire dataset and cross validation in one step `y_pred = cross_val_predict(model, X, y, cv=5)`
- Impute missing values with the mean `X.fillna(X.mean())`.
- Use cross validation to predict precision or recall `recall = cross_val_score(model, X, y, cv=10, scoring="recall")`
- ROC-AUC with cross-validation

```
y_pred = cross_val_predict(logistic, X, y, cv=10, method="predict_proba")
fpr, tpr, _ = roc_curve(y, y_pred[:, 1])
auc_score = auc(fpr, tpr)
```

Unsupervised learning

- show kmeans results in a scatterplot `ax.scatter(X[:,0], X[:,1], c=kmean.labels_, alpha=0.6)`
  - the alpha parameter is the blending value
- Criterions to find k:
  - silhouette score=$(b-a)/max(a,b)$ where b is avg. inter cluster distance and a the avg. intra cluster distance. `silhouette_score(X, labels)`
  - elbow method: look at the sum of all distances to the closest cluster center. Select k at the elbow. `kmeans.inertia_`
- Dimensionality reduction
  - t-stochastic neighbor embedding(t-SNE): gives each datapoint a location in a lower dimensional map with high probability. `X_reduced_tsne = TSNE(n_components=2, init='random', learning_rate='auto', random_state=42).fit_transform(X)`
  - PCA: `X_reduced_pca = PCA(n_components=2).fit(X).transform(X)`
- Remove the mean and scale to unit variance with `preprocessing.StandardScaler().fit_transform(X)`

Handling text

- new lines can be removed by applying `" ".join(txt.split())`
- get a spacy object (tokens) from text `doc = nlp(txt)`
- get sentences as a list `doc.sents`
- get wordtype of a token `[(token.text, token.pos_) for token in doc]`
  - or more detailed with `token.tag_` instead of `token.pos_`
- a list of stopwords is available in spacy `spacy.lang.en.stop_words.STOP_WORDS`
- `token.lemma_` is the lemmatized version of the token
- chuncks consist of a noun and the words describing the noun: `doc.noun_chunks`

- There is a sentiment analyzer for short sentences.

```
analyzer = SentimentIntensityAnalyzer()
vs = analyzer.polarity_scores(example)
print(example, '\n')
print('Negative sentiment:',vs['neg'])
print('Neutral sentiment:',vs['neu'])
print('Positive sentiment:',vs['pos'])
print('Compound sentiment:',vs['compound'])
```

- interpretation
  - positive, negative and neutral sum to one. The compound represents all three values.
- The count vectorizer transforms a collection of text into a matrix of token counts.

```
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(chunks)
```

- The same exists for getting the TF-IDF matrix `TfidfVectorizer()`
- Print the top 20 featurs representing a category:

```
coefs=clf.coef_[0]
top = np.argpartition(coefs, -20)[-20:]

print(np.array(vectorizer.get_feature_names())[top])
```

- we can also represent a sentense as vector using pretrained GloVe `nlp(example).vector`
- topic detection with LDA: `LdaMulticore(corpus=corpus, num_topics=4, id2word=dictionary, workers=6,passes=params['passes'], random_state=params['random_state'])` then show the topics with `model.show_topics(num_words=5)`.
- A dictionary with typical words per category is provided by

```
from empath import Empath
lexicon = Empath()
```

- There is a command to analyze a text `empath_features = lexicon.analyze(doc.text,categories = ["disappointment", "pain", "joy", "beauty", "affection"])`

Networks

- `nx.Graph()` per default, `nx.DiGraph()` for graphs with self-loops, `nx.MultiDiGraph()` for graphs with parallel edges.
- Add nodes `G.add_node(id)`, add edges `G.add_edge(u,v)`
  - can add multiple at once with `G.add_nodes_from(range(2,9))` and `G.add_edges_from([(1,2),(2,3)])`
- Print number of nodes and edges with `nx.info()`

- plot graph with `nx.draw_spring(G, with_labels=True, alpha = 0.8)`
  - or with a circular layout `nx.draw_circular(karateG, with_labels=True, node_color='g', alpha = 0.8)`
- We can create a graph directly from a pandas edgelist with `G =nx.from_pandas_edgelist(edges, 'Source', 'Target', edge_attr=None, create_using= nx.Graph())`.
- Node attributes are added with: `nx.set_node_attributes(G, nodes['Role'].to_dict(), 'Role' )`
- get the sparsity of a network with `nx.density(G)`
- connected components of a graph `nx.connected_components(G)`
  - the boolean query is `nx.is_connected(G)`
- the shortest path between a pair of nodes `nx.shortest_path(quakerG, source="src", target="dst")`
- node induced subraph `subgraph = G.subgraph(node_set)`
- diameter `nx.diameter(G)`
- average shortest paht `nx.average_shortest_path_length(G)`
- global clustering coefficient `nx.transitivity(G)`
- local clustering coefficinet `nx.clustering(G, ['node', ...])`
- get all degrees in a graph `degrees = dict(G.degree(G.nodes()))`
- katz centrality `katz = nx.katz_centrality(G)`
- betweenness centrality `betweenness = nx.betweenness_centrality(G)`
  - choose node colors according to the betweenness centrality:

```
nx.set_node_attributes(G, nx.betweenness_centrality(G), 'betweenness')
pos = nx.spring_layout(G)
ec = nx.draw_networkx(G, pos, nodelist=G.nodes(),
                          node_color=[G.nodes[n]["betweenness"] for n in G.nodes()],
                          node_shape = '.', node_size=1200, font_color="white", font_weig
plt.colorbar(nc)
plt.axis('off')
plt.show()
```

- The Girvan Newman algorithm is implemented as in iterator in nx

```
comp = girvan_newman(G)
it = 0
for communities in itertools.islice(comp, 4):
    it +=1
    print('Iteration', it)
    print(tuple(sorted(c) for c in communities))
```

- The partition with the highest modularity can be optained directly with the Louvain algorithm with `partition = community_louvain.best_partition(quakerG)`.
- We can measure the similarity of connections in a graph with respect to a given attribute trough the attribute assortativity coefficient. `nx.attribute_assortativity_coefficient(G, 'Gender')`