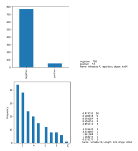
To do List

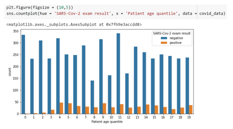
1. What type of data does each category contain?
      i. Can use dataframe.dtypes to see data type
   b. What is the distribution of the data?
      i. Try iterating through the columns and plot the distributions. Consider bar plots for categorical data, and histograms/violin plots for numerical data. Example:



```
negative    768
positive     52
Name: Influenza A, rapid test, dtype: int64
```

```
-0.473031    10
 0.190738     9
 0.694287     9
 0.556955     9
-0.495919     9
             ..
-1.045245     1
-3.334102     1
 1.861604     1
 1.518275     1
-0.976579     1
Name: Hematocrit, Length: 176, dtype: int64
```

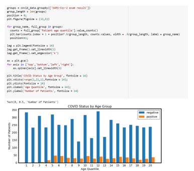
      ii. By looking at the data distribution, are there any columns you want to exclude?
   c. Understanding the data: create some background plots for the presentations.
      i. Seaborn can make your life easier. Example usage:



```
plt.figure(figsize = (10,5))
sns.countplot(hue = 'SARS-Cov-2 exam result', x = 'Patient age quantile', data = covid_data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb9e3accdd8>
```

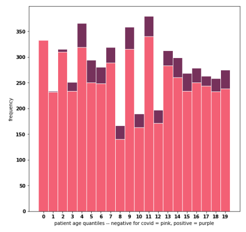
      ii. How to make prettier plots? Example changing borders and fonts with matplotlib:



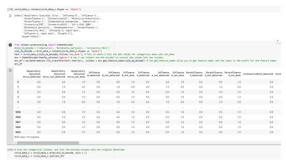
```
groups = covid_data.groupby(['SARS-Cov-2 exam result'])
group_length = len(groups)
position = 0;
plt.figure(figsize = (13,5))

for group_name, full_group in groups:
  counts = full_group['Patient age quantile'].value_counts()
  plt.bar(counts.index + 1 + position*.9/group_length, counts.values, width = .9/group_length, label = group_name)
  position+=1;

leg = plt.legend(fontsize = 14)
leg.get_frame().set_linewidth(1)
leg.get_frame().set_edgecolor('k')

ax = plt.gca()
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(3)

plt.title('COVID Status by Age Group', fontsize = 16)
plt.xticks(range(1,21,1),fontsize = 14);
plt.yticks(fontsize = 14)
plt.xlabel('Age Quantile', fontsize = 14);
plt.ylabel('Number of Patients', fontsize = 14)
```

```
Text(0, 0.5, 'Number of Patients')
```

      iii. Try sns.heatmap(dataframe.corr())
      iv. A stacked bar graph ft. patient age quantiles x covid status!



2. Fill in missing data
   a. Note: you may fill in differently for categorical data vs. numerical data.
      i. For categorical - can use one hot encoding.
         1. Spoiler: here is what I did:



```
[270] covid_data_s.columns[covid_data_s.dtypes == 'object']
```

```
Index(['Respiratory Syncytial Virus', 'Influenza B', 'Influenza A',
       'Parainfluenza 4', 'CoronavirusNL63', 'Rhinovirus/Enterovirus',
       'Parainfluenza 3', 'Chlamydophila pneumoniae', 'Adenovirus',
       'Coronavirus229E', 'CoronavirusOC43', 'Inf A H1N1 2009',
       'Bordetella pertussis', 'Metapneumovirus', 'Parainfluenza 1',
       'Coronavirus HKU1', 'Influenza B, rapid test',
       'Influenza A, rapid test', 'Strepto A'],
      dtype='object')
```

```
from sklearn.preprocessing import OneHotEncoder
#cols_to_encode = ['Adenovirus', 'Bordetella pertussis', 'Coronavirus HKU1']
cols_to_encode = covid_data_s.columns[covid_data_s.dtypes == 'object']
test = covid_data_s[cols_to_encode].fillna('not_done'); # This is where I fill the NaN values for categorical data with not_done
enc = OneHotEncoder(handle_unknown='ignore') # now I use sklearn one hot encoder to convert the column into new columns
enc_df = pd.DataFrame(enc.fit_transform(test).toarray(), columns = enc.get_feature_names(cols_to_encode)) # the get_feature_names allow you to get feature name, and the input is the prefix for the feature names
enc_df
```

```
      Respiratory        Respiratory        Respiratory    Influenza      Influenza  Influenza   Influenza      Influenza  Influenza  Parainfluenza  Parainfluenza  Parainfluenza
        Syncytial          Syncytial          Syncytial   B_detected B_not_detected B_not_done  A_detected A_not_detected A_not_done     4_detected 4_not_detected     4_not_done  CoronavirusNL63_detected  Coro
   Virus_detected Virus_not_detected     Virus_not_done
0             0.0                0.0                1.0          0.0            0.0        1.0         0.0            0.0        1.0            0.0            0.0            1.0                       0.0
1             0.0                1.0                0.0          0.0            1.0        0.0         0.0            1.0        0.0            0.0            1.0            0.0                       0.0
2             0.0                0.0                1.0          0.0            0.0        1.0         0.0            0.0        1.0            0.0            0.0            1.0                       0.0
3             0.0                0.0                1.0          0.0            0.0        1.0         0.0            0.0        1.0            0.0            0.0            1.0                       0.0
4             0.0                1.0                0.0          0.0            1.0        0.0         0.0            1.0        0.0            0.0            1.0            0.0                       0.0
...           ...                ...                ...          ...            ...        ...         ...            ...        ...            ...            ...            ...                       ...
5639          0.0                0.0                1.0          0.0            0.0        1.0         0.0            0.0        1.0            0.0            0.0            1.0                       0.0
5640          0.0                0.0                1.0          0.0            0.0        1.0         0.0            0.0        1.0            0.0            0.0            1.0                       0.0
5641          0.0                0.0                1.0          0.0            0.0        1.0         0.0            0.0        1.0            0.0            0.0            1.0                       0.0
5642          0.0                0.0                1.0          0.0            0.0        1.0         0.0            0.0        1.0            0.0            0.0            1.0                       0.0
5643          0.0                0.0                1.0          0.0            0.0        1.0         0.0            0.0        1.0            0.0            0.0            1.0                       0.0
5644 rows × 57 columns
```

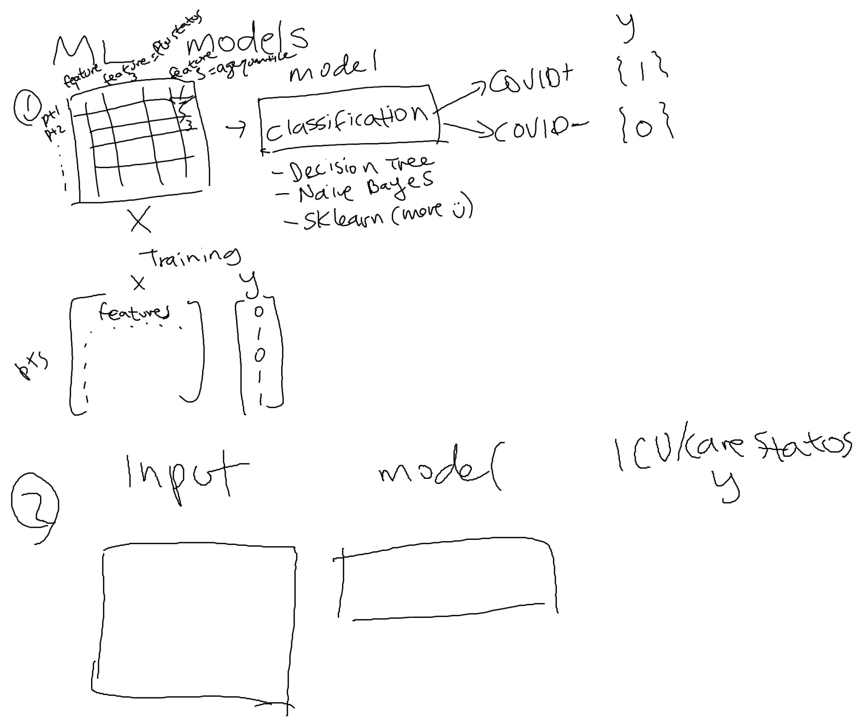
```
[272] # drop the categorical columns, and join the encoded columns with the original dataframe
      covid_data_s = covid_data_s.drop(cols_to_encode, axis = 1)
      covid_data_s = covid_data_s.join(enc_df)
```

      ii. For numerical - can fill NaN with mean, or use sklearn imputer.

3. Input matrix (training data)

Models:

- Classification models for COVID status (pos/neg)
    - Input data → patient data + features (training data)
    - Change Object input data into Int, 0 / 1
    - Output → 1 = COVID+, 0 = COVID-
- ICU/Care Status
    - See distribution of the level of care
    - Input data →
    - Output →

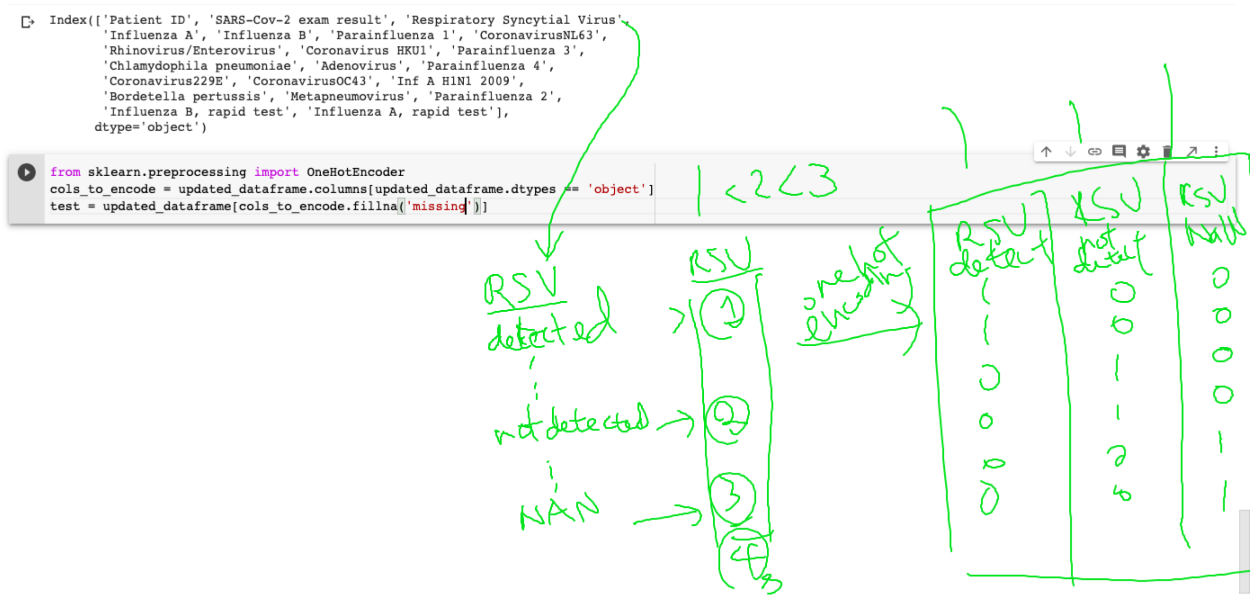


Choose models from:
https://scikit-learn.org/stable/supervised_learning.html

Accomplished Tasks

- 7/20:
    - Set a threshold for exclusion/inclusion of data
        - Threshold: <85% missing
    - Found percent of missing data in all the columns

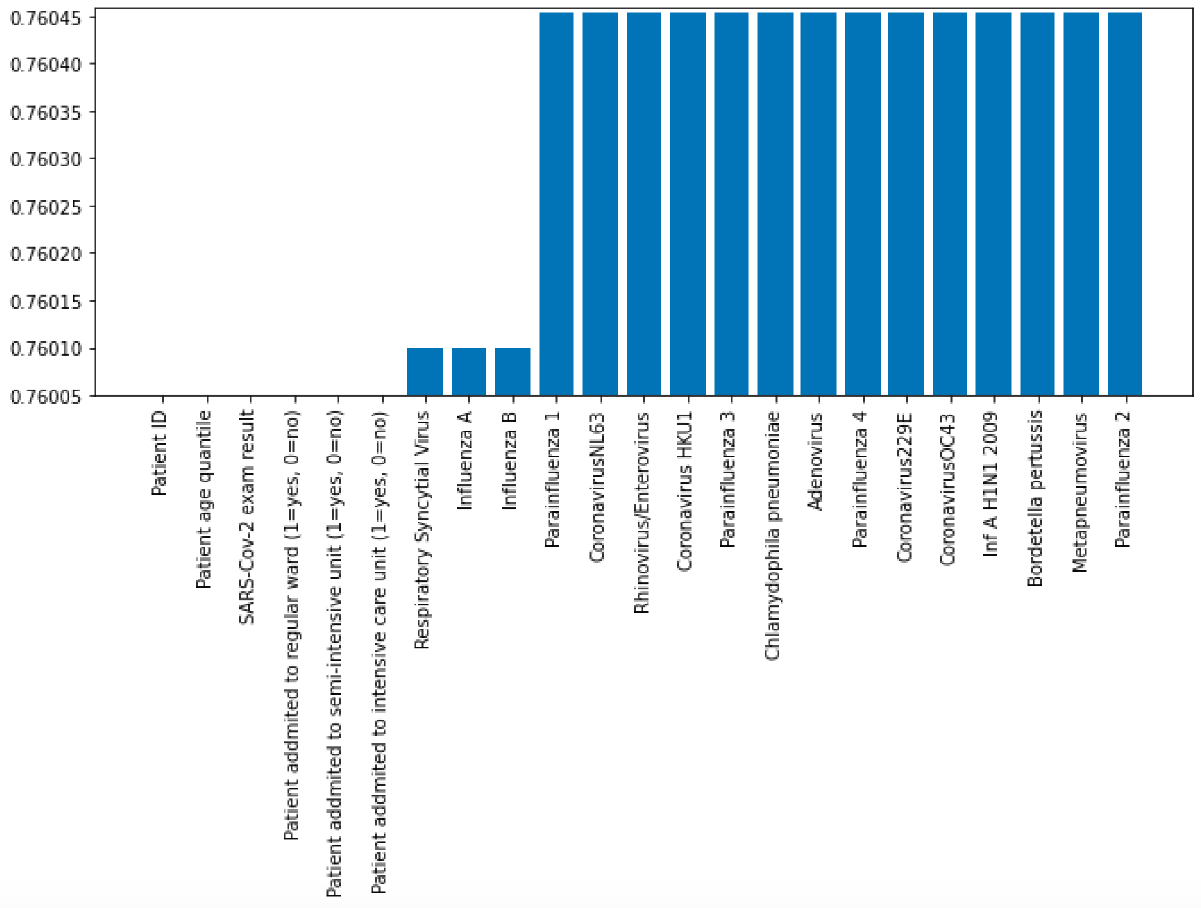
- 7/23:
    - Graphed all categories separately



```
Index(['Patient ID', 'SARS-Cov-2 exam result', 'Respiratory Syncytial Virus',
       'Influenza A', 'Influenza B', 'Parainfluenza 1', 'CoronavirusNL63',
       'Rhinovirus/Enterovirus', 'Coronavirus HKU1', 'Parainfluenza 3',
       'Chlamydophila pneumoniae', 'Adenovirus', 'Parainfluenza 4',
       'Coronavirus229E', 'CoronavirusOC43', 'Inf A H1N1 2009',
       'Bordetella pertussis', 'Metapneumovirus', 'Parainfluenza 2',
       'Influenza B, rapid test', 'Influenza A, rapid test'],
      dtype='object')
```

```python
from sklearn.preprocessing import OneHotEncoder
cols_to_encode = updated_dataframe.columns[updated_dataframe.dtypes == 'object']
test = updated_dataframe[cols_to_encode.fillna('missing')]
```

```
(0.76005, 0.76046)
```



Machine Learning Models

Preprocessing

- Data Cleaning: SimpleImputer, LabelEncoder, OneHotEncoder
- https://scikit-learn.org/stable/modules/preprocessing.html
    - Good to read for overview
- **Scaling/Normalizing** your ‘features’ can be helpful for various reasons such as dealing with batch effects from machine errors. It may also be necessary for certain models to function properly, to avoid bias towards certain features, or for more effective optimization. Wiki article
    - Standard Scaler: Makes data have a mean of 0 and std of 1.

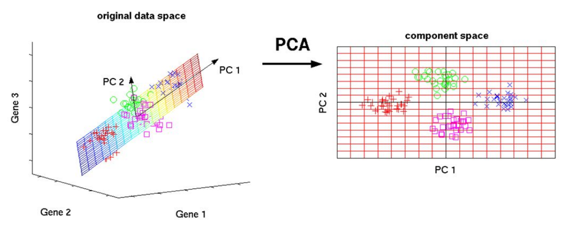
```
from sklearn.preprocessing import StandardScaler
Scaler = StandardScaler()
X = scaler.fit_transform(X)
```

    - Normalizer: Makes your data have a norm (or “length”) of 1.

Dimensionality Reduction

- Using Principal Component Analysis

PCA finds new variables which are linear combinations of the original variables such that in the new space, the data has fewer dimensions.



Think of a data set consisting of points in 3D on the surface of a flat plate held up at an angle. In the original x, y, z axes you need 3 dimensions to represent the data, but with the correct linear transformation, you only need 2.

General idea:

```
from sklearn.decomposition import PCA
pca = PCA()
X_pca = pca.fit_transform(X)
```

```
plt.figure(figsize = (10,8))
sns.lineplot(x = np.arange(pca.n_components_), y =
np.cumsum(pca.explained_variance_ratio_), marker = 'o')
plt.hlines(.80, -10, 90, linestyles = 'dashed', alpha = .5)
plt.ylabel('Cumulative Explained Variance')
plt.xlim([-1,80]); plt.ylim([0,1.1])
plt.xlabel('Number of Components')
```

```
plt.figure(figsize = (10,10))
sns.scatterplot(X_pca[:,0],X_pca[:,1])
```

Clustering? You can try clustering, and color your data with the clusters. Sns.scatterplot takes in a hue parameter to color your data!

Evaluating Models

- **How to export decision trees?**

Method 1:

```
from sklearn import tree
clf = DecisionTreeClassifier(max_depth=10,  class_weight = 'balanced');
clf.fit(X_train, Y_train);

dot_data = tree.export_graphviz(clf, out_file = 'tree.dot',
                     feature_names = feature_names, rounded=True,
                     class_names=['Negative','Positive'], filled=True);
```

Tree.export_graphviz gives you a ".dot" file. If you open the dot file, or the dot_data, you see something like…

```
digraph Tree {\nnode [shape=box, style="filled, rounded", color="black",
fontname=helvetica] ;\nedge [fontname=helvetica] ;\n0 [label="Leukocytes <=
-0.433\\ngini = 0.5\\nsamples = 121\\nvalue = [60.5, 60.5]\\nclass =
Negative", fillcolor="#ffffff"] ;\n1 [label="Eosinophils <= -0.267\\ngini =
0.369\\nsamples = 49\\nvalue = [17.722, 55.0]\\nclass = Positive",
fillcolor="#79bded"] ;\n0 -> 1 [labeldistance=2.5, l...
```

Graphviz is another python package to help you edit and visualize the dot file.
An easy way to convert the .dot file to .png is this line of code on your command line:

```
$ dot -Tpng tree.dot -o tree.png
```

Or more easily in python jupyter notebook, you can use something called 'magic' for command line commands, using **!** .

```
!dot -Tpng tree.dot -o tree.png
```

Method 2: Use sklearn tree.plot_tree method and then use matplotlib to export!

```
fig = plt.figure(figsize=(20,12))
tree.plot_tree(clf, feature_names = feature_names,
               class_names= ['Negative','Positive'],
               filled = True, fontsize = 8);
fig.savefig('tree2.png')
```

EXTRA: you can plot individual decision trees in a random forest with clf.estimators_

- **Better Understanding ROC**
  Wikipedia Article

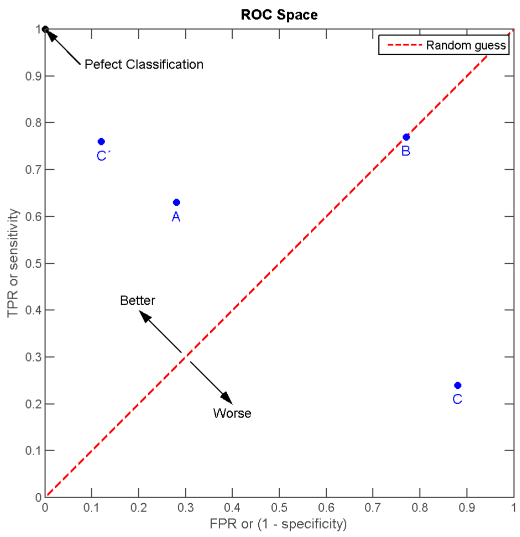
  Let's better understand our confusion matrix and terms used.

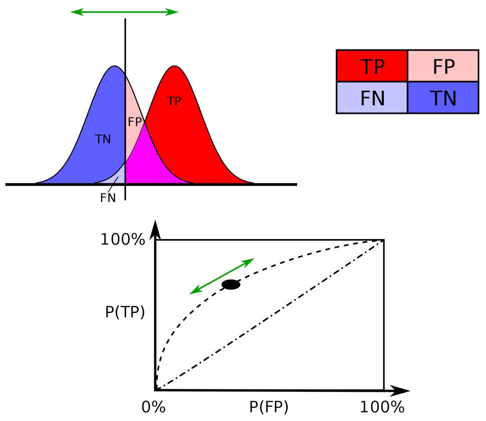
		Predicted Label	
		Positive	**Negative**
True Label	**Positive**	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

  Sensitivity = "Probability of Detection" = True Positive Rate = TP / (TP + FN)
  Specificity = True Negative Rate = TN / (TN + FP)
  False Positive Rate = "False Alarm Rate" = 1 - True Negative Rate = FP / (TN + FP)

  Accuracy = TP + TN / (Population)

  For a Receiver Operating Characteristic curve, you're plotting True Positive Rate vs. False Positive Rate as you vary the threshold.
  On the left, let us say we are calculating the size of a skin lump as being cancer or not. Blue is not cancer, red is cancer. The line is our threshold for size. As we vary our line, we can plot our ROC curve.
  The confusion matrix and accuracy is created at various thresholds - and therefore vary with your threshold. As such, it is possible to have a high AUC, but if you choose a small threshold such that any skin lump is cancer, your accuracy would be low. The AUC is like some average of the accuracy of all the thresholds.

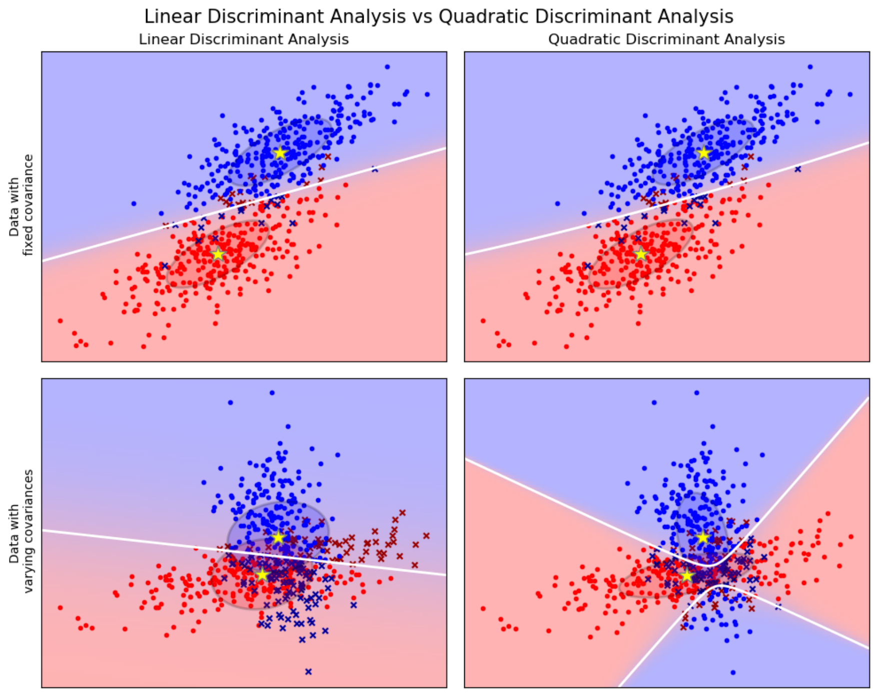




For a ML model, it helps to understand what the model is doing to make a decision between classes and how this decision algorithm works.

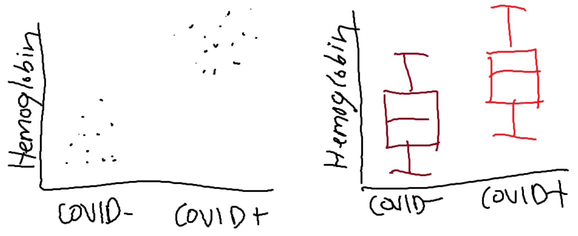
- For probabilistic models, you make have a ‘predict probability’ function, and this probability threshold may be varied in the ROC curve
- For K nearest neighbors, if you look in the documentation, there is a predict probability function, where probability of a class depends upon neighbors in the same class. For decision trees, the probability is based upon correct labels in a leaf.

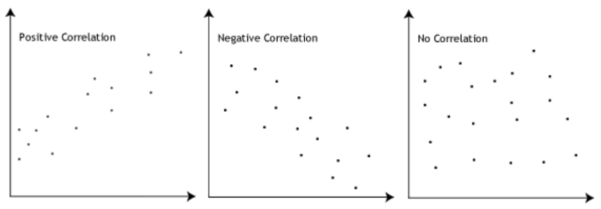
- Quadratic Discriminant Analysis vs. Linear Discriminant Analysis
  https://scikit-learn.org/stable/modules/lda_qda.html



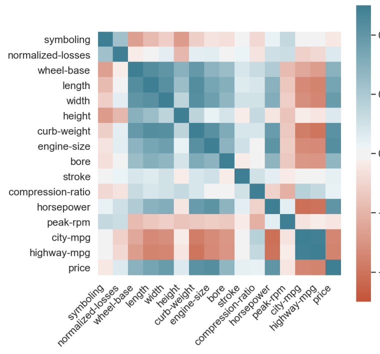
The math:
https://scikit-learn.org/stable/modules/lda_qda.html#mathematical-formulation-of-the-lda-and-qda-classifiers
QDA:
https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html - can also try the reg_param, improves your accuracy/AUC.

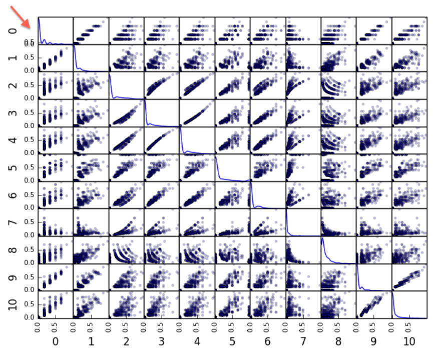
- **Feature Importance:** Good to look at feature importance to figure out what tests are most important in predicting COVID outcome. You can also re-create models using the "top 4" features to suggest an algorithm for physicians to use.
    - Correlation Plots and Box Plots (pre-modelling)
      `Sns.striplot`, `sns.swarmplot`, `sns.violinplot`, `sns.boxplot`





      Correlation of dataframe columns: `dataframe.corr()`
      `sns.heatmap(dataframe.corr())` to visualize as heatmap (left). Just a way to plot correlation of the right (`sns.pairplot()`)

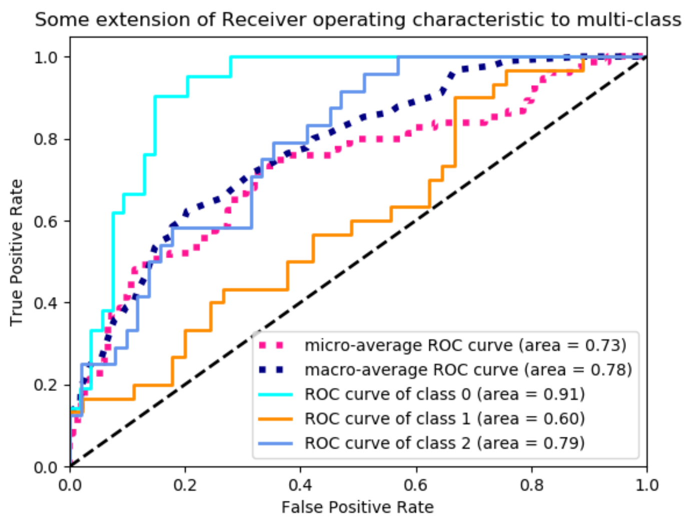




    - Linear models: Like linear regression or ridge classifier. Plot bar graph of coefficient with each of the feature.
    - Decision Trees / Random Forest: feature importance property! Plot it as a bar graph and see what happens.

    - Permutation importance: “permutate” one of the feature columns and see how it affects your score (like how it affects model accuracy) https://scikit-learn.org/stable/modules/generated/sklearn.inspection.permutation_importance.html#sklearn.inspection.permutation_importance

- Multiclass AUC
  https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html



Code: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
Note: y_test has the number of dimensions as # of classes, with 1 at the class label.

```
y = label_binarize(y, classes=[0, 1, 2])
```

```
Y_score = classifier.decision_function(X_test)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(),
y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
```

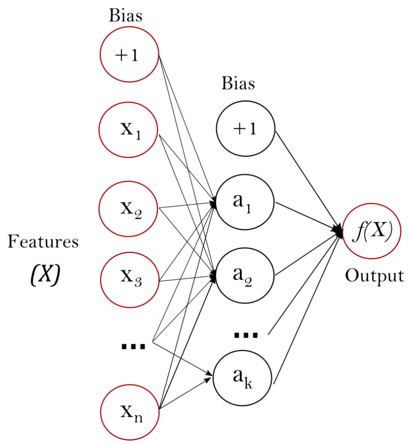
Optimizing Models

- Sklearn tuning parameters and hyperparameters:

Randomized Parameter Optimization is a good option to help you search your parameter space for the best parameters.

Neural Networks

- From sklearn using Multilayer Perceptron: https://scikit-learn.org/stable/modules/neural_networks_supervised.html



MLP limits you to a feed forward fully connected neural network architecture.
Note: MLP is sensitive to feature scaling

MLPClassifier(*hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*)

- Playing with neural networks. You can use the following link to play around with the architecture to modify parameters to the MLP model. https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.33680&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false