Things to do for project.

**These are the packages I used.**

### Load the dataset.

**And find the shape, column, info, head, diagnosis.**

#### Create a correlation map between the variables. Which variables have high correlation (> |0.9|)?

How many people have

## Categorical Features

Notice that the **purpose** column is of categorical type.

That means we need to transform them using dummy variables so sklearn will be able to understand them.

Let's do this in one step using pd.get\_dummies().

Check the statistical summary of numeric apps. Do the variables have values close to each other, or is there large differences?

Among the mean (the first 10) features, which one has the minimum 25% percentile value?

Which mean variable (the first 10 features) has the largest mean value?

Calculate correlation.

Create a correlation heatmap.

Comment on largely correlated variables.

### Prepare data for machine learning

### Train - Test split

* Use sklearn's tran\_test\_split() function to create the split.
* Use random\_state = 55 to create consistent and repeatable train-test splits.
* What is the proportion of cancer classes in train and test sets after splitting? Are they equal or very close to each other?
* What is the importance of stratified sampling?

#### Initiate the logistic regression classifier from the sklearn library as shown below.

* Fit the model
* Make predictions
* Calculate accuracy score. What are the **training** and **testing** accuracies of the model?
* Create confusion matrix
* Either calculate using the confusion matrix, or use the methods in metrics to get the following metrics on the testing set:
  + Recall (Sensitivity)
  + Specificity
  + Precision
  + False Positive Rate
  + F1 Score

Initiate the k-nearest neighbours classifier from the sklearn library with n\_neighbors=1. Keep the remaining parameters with their default values. (No need to specify anything)

* Fit the model
* Make predictions
* Calculate accuracy score. What are the **training** and **testing** accuracies of the model?
* Create confusion matrix
* Either calculate using the confusion matrix, or using the methods in metrics to get the following metrics on the testing set:
  + Recall (Sensitivity)
  + Specificity
  + Precision
  + F1 Score

In logistic regression keeping the other parameters constant try these values for C: C\_list = [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 2, 10, 50, 100, 200, 500, 1000, 2000, 5000, 10000].  
C is the inverse of the regularization parameter $\lambda$. As C increases, the penalty decreases.

* Create a loop and fit the models using each C value.
* Make predictions
* Calculate accuracy scores for **training** and **testing** datasets. Create a list of accuracy results for train and test accuracies. Show these lists.
* Create the validation curve based on each C value and the corresponding train and test accuracies.

Create this curve yourself. Do not use sklearn built-in function.

In K Nearest Neighbors keeping the other parameters constant try these values for K (n\_neighbors): K\_list = [1,3,5,7,9,15,19,25,29,35,39,45,49].

* Create a loop and fit the models using each K value.
* Make predictions
* Calculate accuracy scores for **training** and **testing** datasets. Create a list of accuracy results for train and test accuracies. Show these lists.
* Create the validation curve based on each K value and the corresponding train and test accuracies.

Create this curve yourself. Do not use sklearn built-in function.

The validation curve for logistic regression seems a bit weird. That might have to do with the fact that the data is not normalized.

* Apply standardization to the data.
* In logistic regression keeping the other parameters constant try these values for C: C\_list = [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 2, 10, 50, 100, 200, 500, 1000, 2000, 5000, 10000].
* Plot validation curves. What do you see?
* Which value would you choose for C in logistic regression?

After choosing your best C value, let's check the learning curve.

* Create a logistic regression model of your chosen C value.
* Starting from 10 data points in the training set, increase your training set size by 1 points in each iteration, fit the model, get the accuracy scores for both the current training and test sets.
* Increase the training set size consecutively. Do not randomly select data points. (X\_train\_ss[0:N,:] where N = {1,2,3,...,len(X\_train\_ss)})
* Plot the learning curve. What do you see? Comment on your finding.

## Decision Tree Model

Let's start by training a single decision tree first!

1. Default decision tree:
   * Initiate the Decision Tree classifier from the sklearn library. Keep all parameters with their default values. (No need to specify anything)
   * Fit the model
   * Make predictions
   * Calculate accuracy score. What are the **training** and **validation** accuracies of the model?
   * Create confusion matrix
   * Provide the performance metrics for the **validation** set.
2. Using the training data perform hyper-parameter tuning. Use an appropriate metric for scoring (f1 is generally good).
   * Try several values for the min\_samples\_split hyper-parameter and apply cross validation (GridSearchCV) to find the best value.
     + Using the best model provide the performance metrics for the validation set.
   * Try several values for the max\_depth hyper-parameter and apply cross validation (GridSearchCV) to find the best value.
     + Using the best model provide the performance metrics for the validation set.
   * Perform grid search (GridSearchCV) where you check for min\_samples\_split and max\_depth.
     + Using the best model provide the performance metrics for the validation set.

## Random Forest model

1. Default Random Forest Classifier:
   * Initiate the Random Forest Classifier from the sklearn library. Keep all parameters with their default values. (No need to specify anything)
   * Fit the model
   * Make predictions
   * Calculate accuracy score. What are the **training** and **validation** accuracies of the model?
   * Create confusion matrix
   * Provide the performance metrics for the **validation** set.
2. Using the training data perform hyper-parameter tuning. Use an appropriate metric for scoring (f1 is generally good).
   * Perform grid search (GridSearchCV) where you check for min\_samples\_split, max\_depth, and n\_estimators.
     + Using the best model provide the performance metrics for the validation set.
3. Create a chart showing the feature importances based on the best model obtained.

## Adaboost model

1. Default Adaboost classifier:
   * Initiate the Adaboost classifier from the sklearn library. Keep all parameters with their default values. (No need to specify anything)
   * Fit the model
   * Make predictions
   * Calculate accuracy score. What are the **training** and **validation** accuracies of the model?
   * Create confusion matrix
   * Provide the performance metrics for the **validation** set.
2. Using the training data perform hyper-parameter tuning. Use an appropriate metric for scoring (f1 is generally good).
   * Perform grid search (GridSearchCV) where you check for learning\_rate, and n\_estimators.
     + Using the best model provide the performance metrics for the validation set.
3. Create a chart showing the feature importances based on the best model obtained.

## Gradient Boosting Machine

1. Default Gradient Boosting classifier:
   * Initiate the Gradient Boosting classifier from the sklearn library. Keep all parameters with their default values. (No need to specify anything)
   * Fit the model
   * Make predictions
   * Calculate accuracy score. What are the **training** and **validation** accuracies of the model?
   * Create confusion matrix
   * Provide the performance metrics for the **validation** set.
2. Using the training data perform hyper-parameter tuning. Use an appropriate metric for scoring (f1 is generally good).
   * Perform grid search (GridSearchCV) where you check for learning\_rate, and n\_estimators.
     + Using the best model provide the performance metrics for the validation set.
3. Create a chart showing the feature importances based on the best model obtained.

Apply Linear Regression. Predict CO emission using other variables.

* + What is the model $R^2$ on training data?
  + What is the $R^2$ on validation data?
  + What is the Root Mean Squared Error (RMSE) on training data?
  + What is the RMSE on validation data?
* $R^2$ higher is better. ($R^2$ is % variation in data being explained)
* RMSE lower is better. (RMSE (Root Mean Squared Error) is error, meaning the difference between predictions and actual data)

Hint:

* You can use the r2\_score and mean\_squared\_error methods in sklearn.metrics.
* You need to take the square root of Mean Squared Error to get to RMSE. (MSE $\rightarrow$ square root $\rightarrow$ RMSE)

Apply Random Forest Regression with 100 trees. Predict CO emission using other variables.

* + What is the model $R^2$?
  + What is the $R^2$ on validation data?
  + What is the RMSE on training data?
  + What is the RMSE on validation data?
* Is training and validation score close? If not what might be the problem?
* If there is a large gap, try to decrease the gap by changing some hyperparameters of the Random Forest model. (In this analysis, any difference more than 0.10 $R^2$ difference can be assumed to be large.)
  + Once you reach a difference of less than or close to 0.10 between training $R^2$ and validation $R^2$ you can stop.
  + During this process your training score will probably decrease, and that is OK.

Apply Support Vector Regression. Predict CO emission using other variables.

* + What is the model $R^2$?
  + What is the $R^2$ on validation data?
  + What is the RMSE on training data?
  + What is the RMSE on validation data?
* Do this for each of the models listed below using the default parameters:
  + **Linear kernel SVR**
  + **Polynomial kernel SVR**
  + **RBF kernel SVR**

Take RBF Kernel SVM and perform hyperparameter tuning. See if you can increase the $R^2$ score.

* + Use:  
    'C': [1, 10, 100, 1000] 'gamma': [0.001, 0.01, 0.1, 1]
  + What is the model $R^2$?
  + What is the $R^2$ on validation data?
  + What is the RMSE on training data?
  + What is the RMSE on validation data?

Apply PCA and transform the data.

* + For PCA we need to decide on the number of components. Take sufficient number of components based on variance explained
    - generally it is chosen as the number of components that can explain 90% of the variance
    - or cross-validation accuracy using an ML model.
* Use the PCA transformed data to perform regression. Use the models:
  + Linear Regression
  + Random Forest Regression (Optional to do hyper parameter tuning)
  + RBF Kernel Support Vector Regression (Optional to do hyper parameter tuning)
* For each model:
  + What is the model $R^2$?
  + What is the $R^2$ on validation data?
  + What is the RMSE on training data?
  + What is the RMSE on validation data?
* Compare these models with their counterparts obtained before. For example: Compare PCA RBF SVR with the RBF SVR model from before. Do you observe any change in performance? Please explain.

### Fit RBF Kernel SVM Classifier

#### 1. Standardized training data

* Using the standardized training data perform grid search:
  + Find the best combination of the C and gamma hyper-parameters.
* For the best model:
  + Either using the confusion matrix or the methods in sklearn.metrics get the following metrics on the validation set:
  + Create confusion matrix
  + Recall (Sensitivity)
  + Specificity
  + Precision
  + Balanced Accuracy
  + F1 Score

#### 2. PCA transformed training data

* Using the **pca** transformed training data X\_train\_pca2 perform grid search:
  + Find the best combination of the C and gamma hyper-parameters.
* For the best model:
  + Either using the confusion matrix or the methods in sklearn.metrics get the following metrics on the validation set X\_val\_pca2:
    - Create confusion matrix
    - Recall (Sensitivity)
    - Specificity
    - Precision
    - Balanced Accuracy
    - F1 Score

For regression:

* + Think of other ways of improving the regression performance and list at least two possibilities.
  + You can use any of the datasets we have been using (standardized data, PCA transformed data), or re-create your own set from the original.
  + Implement both of the methods you proposed.
  + Is the validation performance increasing? For performance comparison you can use any regression performance metric that you deem to be appropriate.
  + Optional: Apply different methods, ideas as you see fit and see if the validation performance is improving.