Question & Answer

Intro Questions:

- What is the difference between supervised & unsupervised learning?
  - Supervised uses labeled data & is trained, unsupervised does not & is untrained
- How do you ensure your model is not overfitting?
  - Reduce noise by using less variables
- How do you ensure your model is not underfitting?
  - Increase features, not data
- Why do we split the data into training and testing sets?
  - To ensure our model performs well on data it is not being trained on
- How much data should you be adding to your training set?
  - 75% training 25% testing is a good rule of thumb
- What are the steps to building a model?
  1. Selecting & Preparing a Dataset
  2. Choosing an Algorithm to Run on the Datasets
  3. Training the Algorithm
  4. Using & Improving the Model
- Using the fake dataset called "data" simulate how you would read the data.

```
#read and store the data
dummy_data = pd.read_csv(data)

#print data summary
dummy_data.describe()
```

- Simulate how you would split the test and train data.

```
from sklearn.model_selection import train_test_split

train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 1)
```

- Simulate how you would define a decision tree model.

```
from sklearn.tree import DecisionTreeRegressor

#define
decsion_model = DecisionTreeRegressor(random_state = 1)
```

- Simulate how you would define a random forest model.

```
#instead of importing DecisionTreeRegressor, let's import RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor

#define
forest_model = RandomForestRegressor(random_state = 1)
```

- Simulate how you would fit your model.

```
my_model.fit(train_X,train_y)
```

- Simulate how you would predict your results.

```
predictions = my_model.predict(test_X)
```

- Simulate how you would see the mean absolute error of your model.

```
from sklearn.metrics import mean_absolute_error

mean_absolute_error(val_y, predictions)
```

Intermediate Questions:

- What are the ways you can handle missing values?
  - Drop Columns with Missing Values - If there is a column that contains even a singular missing value it is dropped from the DataFrame entirely
  - Imputation - Filling the missing values with a number (often the average of other values in the column)
- What are the ways you can handle categorical values?
  - Drop Categorical Variables - Just like when dropping missing values, when there is a categorical variable it's column is dropped
    - This only works when the categorical information is not useful
  - Label Encoding - Assigns an integer to each unique value
    - Example: 'Strongly Agree' = 5, 'Agree' = 4, 'Neutral' = 3, 'Disagree' = 2, 'Strongly Disagree' = 1
    - Does not work well if the values do not fall into clear ordering
  - One-Hot Encoding - Creates new columns that show the presence/absence of each categorical value in the original data
    - Example: 'Strongly Agree', 'Agree', 'Neutral', 'Disagree', and 'Strongly Disagree' would become columns and each time the specific variable was used in the original data, there would be a 1 in the corresponding row in that column and any time the variable was not used a 0 would be put in the row
    - Does not work well with datasets that have large amounts of different categorical values
- What are the benefits of pipelines?
  - Producing cleaner code - You do not need to manually keep track of training/testing data at each step
  - Producing fewer bugs - Fewer chances to forget or mess up applying a step during preprocessing
  - Easier to productionize - Models can be hard to move from a prototype to something large scale. Pipelines can help with the transition
  - More options of model validation - Ex. Cross validation
- When do you use cross-validation?
  - Small Datasets - When it is not a big deal to have extra computational costs, use cross-validation
  - Large Datasets - A single validation set is good, it will make your code run faster and a large set will allow you to take more data from the testing set without causing any major disruptions in model accuracy
- What are the main types of data leakage?
  - Target Leakage - When your predictors include data that is not available during prediction time

        - Need to look at the chronology of the data - any variable created after the target value should be excluded from the dataset
    - Train-Test Contamination - When the validation data affects the preprocessing behavior
        - Ex. fitting an imputer before calling train_test_split. The validation scores would be great, but the performance in testing could be awful
        - Make sure if using a train-test split that all validation data is excluding from an instance of fitting

- What are the steps for the gradient boosting method?
    - Current ensemble generates a prediction by adding the predictions from all the models in the ensemble
    - A loss function is calculated using the predictions
    - The loss function is used to fit a new model which is added to the ensemble - in hopes that adding the new model will reduce the loss
    - Add the new model
    - Repeat
- What are the steps for the K-Nearest Neighbor algorithm?
    1. Find a dataset and load the training and testing data
    2. Choose the value of K - the number of nearest neighbors used as the majority in the "voting" process
    3. For each point:
        a. Calculate the distance between test data and each row of training data
        b. Sort them based on distance value (ascending)
        c. Chose the top K rows from the sorted array
        d. Assign a class to each test point based on most frequent class of rows

- How is the Support Vector Machine Algorithm created?

  The way SVM works is that a line is created between two separate classes so that all data points that are one one side of the line will correspond with one category, and all of the points on the other side of the line will be for the other category.
- Show the steps for dropping missing values

```
col_with_missing_values = [col for col in x_train.columns if x_train[col].isnull().any()]
reduced_X_train = X_train.drop(cols_with_missing_values, axis = 1)
reducted_X_test = X_test.drop(cols_with_missing_values, axis = 1)
```

- Show the steps for using imputation for missing values

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer()
imputed_X_train = pd.DataFrame(imputer.fit_transform(X_train))
imputed_X_test = pd.DataFrame(imputer.transform(X_test))

#Imputation removes column names, so we have to add those back in
imputed_X_train.columns = X_train.columns
imputed_X_test.columns = X_test.columns
```

- Show the steps for dropping categorical values

```
#Drop values
reduced_X_train = X_train.select_dtypes(exclude = ['object'])
reduced_X_test = X_test.select_dtypes(exclude = ['object'])
```

- Show the steps for handling categorical values using label encoding

```
#Label Encoding
from sklearn.preprocessing import LabelEncoder

#First you have to check if the columns can be safely label encoded
#Grab all categorical columns
object_columns = [col for col in X_train.columns if X_train[col].dtype == "object"]

#Grab all safe columns
good_labels = [col for col in object_columns if set(X_train[col]) == set(X_test[col])]

#Drop unsafe columns
bad_labels = list(set(object_columns) - set(good_labels))
label_X_train = X_train.drop(bad_labels, axis = 1)
label_X_test = X_test.drop(bad_labels, axis = 1)

#Apply label encoder
label_encoder = LabelEncoder()
for col in set(good_label):
  label_X_train[col] = label_encoder.fit_transform(X_train[col])
  label_X_test[col] = label_encoder.transform(X_test[col])
```

- Show the steps for handling categorical values using one-hot encoding

```
#One-Hot Encoding
from sklearn.preprocessing import OneHotEncoder

#Check for cardinality of object columns (number of unique entries of a categorical variable)
low_cardinality = [col for col in object_columns if X_train[col].nunique() < 10]

#Apply One-Hot Encoder
OH_encoder = OneHotEncoder(handle_unknown = 'ignore', sparse = False)
OH_train_columns = pd.DataFrame(OH_encoder.fit_transform(X_train[low_cardinality]))
OH_test_columns = pd.DataFrame(OH_encoder.transform(X_test[low_cardinality]))

#Re-input index
OH_train_columns.index = X_train.index
OH_test_colums.index = X_test.index

#Remove categorical variables from original datasets
noCat_X_train = X_train.drop(object_columns, axis = 1)
noCat_X_test = X_test.drop(object_columns, axis = 1)

#Add one-hot encoded columns to original columns
OH_X_train = pd.concat([noCat_X_train, OH_train_columns], axis = 1)
OH_X_test = pd.concat([noCat_X_test, OH_test_columns], axis = 1)
```

- Show the steps for creating a pipeline

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestRegressor

numerical_transformation = SimpleImputer(strategy = 'constant')
categorical_transformation = Pipeline(steps = [('imputer', SimpleImputer(strategy = 'most_frequent')), ('onehot', OneHotEncoder(handle_

preprocessed = ColumnTransformer(transformers = [('num', numerical_transformation,
numerical_columns), ('cat', categorical_transformation, categorical_columns)])

my_model = RandomForestRegressor(n_estimators = 100, random_state = 0)

#Bundle preprocessing and modeling code into a pipeline
pipeline_bundle = Pipeline(steps = [('preprocessor', preprocessed), ('model', model)])
```

```
#Preprocess training data & fit model
pipeline_bundle.fit(X_train, y_train)

#Preprocess validation data & get predictions
predictions = pipeline_bundle.fit(X_val)

print('MAE:', mean_absolute_error(y_val, predictions))
```

- how the code for cross-validation

```
from sklearn.model_selection import cross_val_score
#sklearn calculates a negative MAE so multiply by -1
scores = -1 * cross_val_score(pipeline_bundle, X, y, cv = 5, scoring = 'neg_mean_absolute_error')
```

- Simulate how you would define a gradient boost model

```
from xgboost import XGBRegressor

#Define
boost_model = XGBRegressor(random_state = 0)

#Fit
boost_model.fit(X_train, y_train)

#Get predictions
prediction = boost_model.predict(X_val)
mae_boost = mean_absolute_error(prediction, y_val)
```

- Using a fake dataset called "data" , whos first value is the decision and rest of values are variables, simulate how to create a K-NN model

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error

X_full = pd.read_csv(data.csv)
X_full.head()

#Dropping Columns
X_full = X_full.drop(['id', 'Unnamed: 32'], axis = 1)

#Preprocessing (Kinda)
y = X_full.iloc[:,0]
X = X_full.iloc[:,1:31]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

#Data Scaling
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

#Training
knn_classifer = KNeighborsClassifier(n_neighbors = 5)
knn_classifer.fit(X_train, y_train)

#Get Predictions
predictions = knn_classifer.predict(X_test)

#Print results
result = confusion_matrix(y_test, predictions)
print("Confusion Matrix: ")
print(result)

classification_result = classification_report(y_test, predictions)
print("Classification Report:")
print(classification_result)
```

```
accuracy_result = accuracy_score(y_test, predictions)
print("Accuracy:")
print(accuracy_result)
```

- Using a fake dataset called "data", whos first value is the decision and rest of values are variables, simulate how to create a SVM model

```
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score

#Read data
X_full = pd.read_csv(data.csv)
X_full.head()

#Dropping Columns
X_full = X_full.drop(['id', 'Unnamed: 32'], axis = 1)

#Preprocessing
y = X_full.iloc[:,0] #First value = decision
X = X_full.iloc[:,1:31] #Rest of values = variables

#Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

#Define Model
svm_model = svm.SVC(kernel = 'linear')

#Train Model
svm_model.fit(X_train, y_train)

#Predict
predictions = svm_model.predict(X_test)
accuracy_result = accuracy_score(y_test, predictions)
print(accuracy_result)
```