**DATA 624 Project 2 Python Supplement**

[**Executive Summary**](#_sdbnmoqqw5s0) **1**

[**Exploratory Data Analysis**](#_9bk2x6ehgfbr) **2**

[**Data Preparation**](#_er6gba7ak0jm) **3**

[**Model Building**](#_3z7cv3eqhpw1) **4**

[**Model Selection**](#_as6tww5fkkt) **5**

[**Prediction**](#_6e1pe8xh56ui) **7**

[**Conclusion**](#_s7ob5ck6mhoy) **7**

# Executive Summary

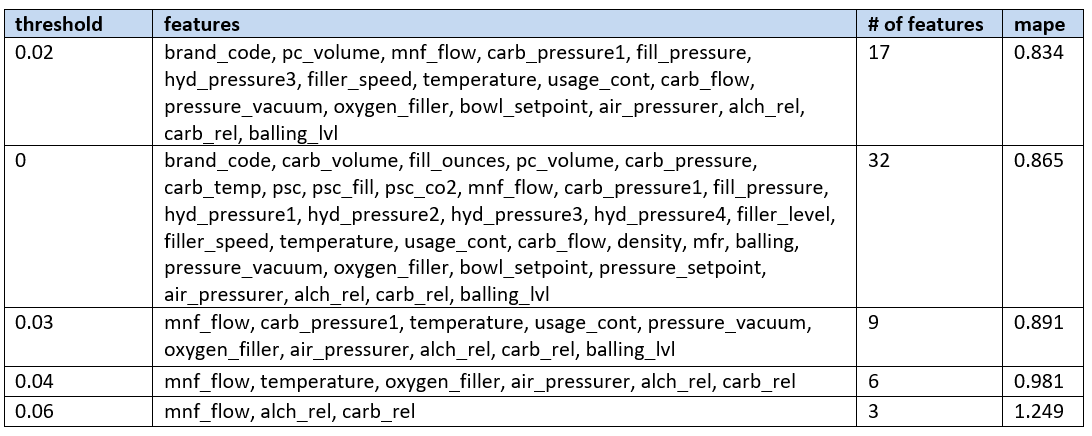
The members of Group 3 voted to accept Professor Burk’s offer of potential extra credit for addressing Project 2’s requirements via both R and Python. We decided that the work completed in Python is best shared in a manner that supplements the write-up in R, rather than duplicates it.

The “Python Experience” for this project entailed the use of 6 algorithms (Random Forest, kNN, Decision Tree, GBM, SVM, and MARS). Random Forest (RF) presented the best results, with a surprisingly low Mean Average Percent Error (MAPE) of around 1%.

Using RF’s ability to assign an importance to each feature, we created six distinct RF models as shown in the table below. The RF model with 17 features performed best with a MAPE of 0.83%. However, it’s very interesting to note that a much simpler model with only 3 features (mnf\_flow, alch\_rel, and carb\_rel) has a MAPE of 1.25%, a degradation of less than a half-percent.

Given this low degradation, we recommend considering the simpler 3 feature model for deployment to Production, versus a more complex 17 feature model.

*Summary of Random Forest Models:*



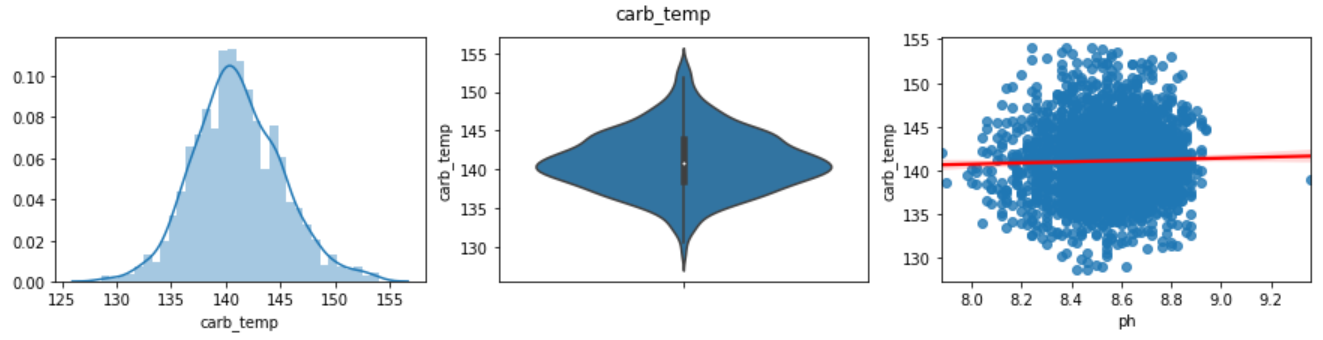
# Exploratory Data Analysis

We explored the data by compiling descriptive statistics and also several visualizations for each feature…

**Descriptive Statistics:**

* 32 features, plus the label (PH) that we were asked to predict
* 2541 instances (rows)
* A 70/30 train/test split was created, so 1779 training rows and 762 testing rows
* The Brand code…
  + Is represented across 5 categories (A, B, C, D, and unknown)
  + This data was skewed heavily towards category B at 1,240 values. This turned out to not be relevant as Brand Code is not an important feature to predict PH
* The PH value is null for 4 of the rows
* There are 438 rows where at least one column was blank (null). These null values are addressed during Data Preparation.

|  |
| --- |
| **#Imports import numpy as np import pandas as pd from IPython.core.display import display, HTML import matplotlib.pyplot as plt import seaborn as sns from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestRegressor from sklearn.feature\_selection import SelectFromModel from sklearn.metrics import mean\_squared\_error pd.set\_option('Display.max\_columns', None) pd.set\_option('Display.max\_rows', None) data = pd.read\_csv('https://raw.githubusercontent.com/KevinJpotter/msds/main/624/Projects/Project2/StudentData%20-%20TO%20MODEL.csv')  def visuals(data):   display(HTML("<style>div.output\_scroll { height: 30em; }</style>"))   sorted\_colnames = list(data.columns.sort\_values())  for feature\_name in sorted\_colnames:  fig, axes = plt.subplots(1, 3, figsize=(15, 3))  fig.suptitle(feature\_name)  sns.distplot(data[feature\_name],ax=axes[0])  sns.violinplot(data[feature\_name], orient='v', ax=axes[1])  sns.regplot('ph', feature\_name, data, line\_kws={'color': 'red'}, ax=axes[2])  plt.show() visuals(data)** |



For each feature, we created a histogram, violin plot, and regression plot (feature ~ label). Above is a sample of what this visual looks like and the notebook contains all of the features’ plots. These charts were used to assess near-normal and skewed distributions, and also to get a visual view on if the data for a given feature is a closely distributed range or has a significant number of outliers. After plotting all variables we don’t identify any explicit linear relationships which lets us know there will be a difficult to decipher the variables that will lead to the best model performance.

# Data Preparation

|  |
| --- |
| #Function to clean the data def clean\_data(data, model\_or\_pred):  #Rename columns to lower case  data.columns = data.columns.str.lower().str.replace(' ','\_')   #Replace letter brand\_codes (A,B,C,D) with integers and nulls with 5  unknown\_brand\_codes\_count = data.brand\_code.isnull().sum()  data['brand\_code'] = data.brand\_code.fillna('unknown')  #data['brand\_code2'] = pd.factorize(data['brand\_code'])[0] + 1  data['brand\_code'] = data['brand\_code'].replace(to\_replace=['A', 'B', 'C', 'D', 'unknown'], value=[1, 2, 3, 4, 5])  ph\_isnull\_count = data.ph.isnull().sum()  #Remove rows that have a PH of null in the model data, but skip this step for the final predictions data  if model\_or\_pred == 'model':  data = data[data.ph.notnull()]  #Remove rows that are all nulls  row\_all\_null\_values\_count = data.isnull().all(axis=1).sum()  data = data.dropna(how='all', axis=0)  #count rows with at least one null  row\_at\_least\_one\_null\_count = data.isnull().any(axis=1).sum()   return(data) data = clean\_data(data, 'model') |

The dataset for this project was among the cleanest our team has ever worked with **(Thank you!!).**

All the features except Brand Code were already numeric. Brand Code contained types of “A”,”B”,”C” and “D”. We converted this to corresponding integers of 1, 2, 3, 4, and 5 for cases where there was a null value.

There are 438 rows in the model data with at least one null value. Based on a visual review of each feature’s distribution, features with a near-normal distribution had nulls replaced with the mean value and features with skewed distributions were addressed via replacing nulls with the median value. The above function takes in a raw dataset and preprocesses it in a way that is suitable for modeling. Many algorithms do not accept null values so it’s important we impute those values before passing those values to a model.

# Model Building

We built the below models and the best performing model was selected based on MAPE values

1. GBM (Gradient Boosting Regressor)
2. kNN (k-nearest Neighbors)
3. SVM (Support Vector Machine )
4. Decision Tree
5. MARS
6. Random Forest (Multiple Thresholds)

**Gradient Boosting Regressor (GBM)**

GBM, short for “Gradient Boosting Machine”, is introduced by Friedman in 2001. It is also known as MART (Multiple Additive Regression Trees) and GBRT (Gradient Boosted Regression Trees). GBM constructs a forward stage-wise additive model by implementing gradient descent in function space.

|  |
| --- |
| from sklearn.ensemble import GradientBoostingRegressor model = GradientBoostingRegressor(max\_depth=10) model.fit(X\_train, y\_train) importance = model.feature\_importances\_ print(len(X\_train.columns)) df = pd.DataFrame(columns=['feature','importance']) cols=X\_train.columns[0:32] rows\_list = [] for i,v in enumerate(importance):  dic1={}  feature\_res=cols[i-1]  importance\_res=v  dic1.update({'feature':feature\_res,'importance':v})  rows\_list.append(dic1) df=df.append(rows\_list)  y\_gbm = model.predict(X\_test)  models\_gbm\_mape = round(mape(y\_test, y\_gbm),3)  print('MAPE for the GBM model is', models\_gbm\_mape, '%') |

MAPE for the GBM model is 0.852 %

**k-Nearest Neighbors (kNN)**

The k-Nearest Neighbors algorithm or KNN for short is a very simple technique. The entire training dataset is stored. When a prediction is required, the k-most similar records to a new record from the training dataset are then located. From these neighbors, a summarized prediction is made.

|  |
| --- |
| **from** sklearn**.**neighbors **import** KNeighborsRegressor  neigh **=** KNeighborsRegressor**(**n\_neighbors**=**10**)**  neigh**.**fit**(**X\_train**,** y\_train**)**  y\_knn **=** neigh**.**predict**(**X\_test**)**  models\_knn\_mape **=** **round(**mape**(**y\_test**,** y\_knn**),**3**)**  **print(**'MAPE for the kNN model is'**,** models\_knn\_mape**,** '%'**)** |

MAPE for the kNN model is 1.202 %

**Support Vector Machine (SVM)**

A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labeled training data for each category, they’re able to categorize new text. Compared to newer algorithms like neural networks, they have two main advantages: higher speed and better performance with a limited number of samples (in the thousands). This makes the algorithm very suitable for text classification problems, where it’s common to have access to a dataset of at most a couple of thousands of tagged samples.

|  |
| --- |
| **from** sklearn**.**metrics **import** mean\_squared\_error  **from** sklearn**.**svm **import** SVR  **from** sklearn**.**model\_selection **import** cross\_val\_score  regressor **=** SVR**(**kernel**=**'linear'**)**  regressor**.**fit**(**X\_train**,**y\_train**)**  cols**=**X\_train**.**columns**[**0**:**32**]**  cols  **print(**regressor**.**coef\_**)**  imp**,**names**=**regressor**.**coef\_**,** cols  #print(imp[0])  df **=** pd**.**DataFrame**(**columns**=[**'feature'**,**'importance'**])**  rows\_list**=[]**  j**=**0  **for** i **in** imp**[**0**]:**  v**=**names**[**j**]**  dic1**={}**  dic1**.**update**({**'feature'**:**i**,**'importance'**:**v**})**  rows\_list**.**append**(**dic1**)**  j**=**j**+**1  df**=**df**.**append**(**rows\_list**)**  #print(df)  #print(out)  df\_result\_1**=**df**.**sort\_values**(**by**=**'importance'**,** ascending**=False).**head**(**10**)**  **print(**df\_result\_1**)**  y\_svm **=** regressor**.**predict**(**X\_test**)**  models\_svm\_mape **=** **round(**mape**(**y\_test**,** y\_svm**),**3**)**  **print(**'MAPE for the SVM model is'**,** models\_svm\_mape**,** '%'**)** |

MAPE for the SVM model is 27.606 %

**Support Vector Machine (SVM)**

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

|  |
| --- |
| **from** sklearn**.**tree **import** DecisionTreeClassifier**,**DecisionTreeRegressor  **from** sklearn **import** tree  clf **=** tree**.**DecisionTreeRegressor**()**  clf **=** clf**.**fit**(**X\_train**,** y\_train**)**  # get importance  importance **=** clf**.**feature\_importances\_  df **=** pd**.**DataFrame**(**columns**=[**'feature'**,**'importance'**])**  cols**=**X\_train**.**columns**[**0**:**32**]**  cols  rows\_list **=** **[]**  **for** i**,**v **in** **enumerate(**importance**):**  dic1**={}**  feature\_res**=**cols**[**i**-**1**]**  importance\_res**=**v  dic1**.**update**({**'feature'**:**feature\_res**,**'importance'**:**v**})**  rows\_list**.**append**(**dic1**)**  df**=**df**.**append**(**rows\_list**)**  df\_result\_1**=**df**.**sort\_values**(**by**=**'importance'**,** ascending**=False).**head**(**10**)**  **print(**df\_result\_1**)**  y\_dt **=** clf**.**predict**(**X\_test**)**  models\_dt\_mape **=** **round(**mape**(**y\_test**,** y\_dt**),**3**)**  **print(**'MAPE for the Decision Tree model is'**,** models\_dt\_mape**,** '%'**)** |

MAPE for the Decision Tree model is 1.127 %

**Multivariate Adaptive Regression Splines (MARS)**

MARS model of individual behavior is a model that seeks to elaborate individual behavior as a result of internal and external factors or influences combined together. The name itself is an acronym for individual Motivation, Abilities, Role Perception and Situational Factors.

|  |
| --- |
| **from** pyearth **import** Earth  model **=** Earth**(**feature\_importance\_type**=(**'rss'**,** 'gcv'**,** 'nb\_subsets'**))**  model**.**fit**(**X\_train**,**y\_train**)**  **print(**model**.**summary\_feature\_importances**(**sort\_by**=**'gcv'**))**  y\_mars **=** model**.**predict**(**X\_test**)**  models\_mars\_mape **=** **round(**mape**(**y\_test**,** y\_mars**),**3**)**  **print(**'MAPE for the MARS model is'**,** models\_mars\_mape**,** '%'**)** |

MAPE for the Decision Tree model is 1.226 %

**Random Forest (RF)**

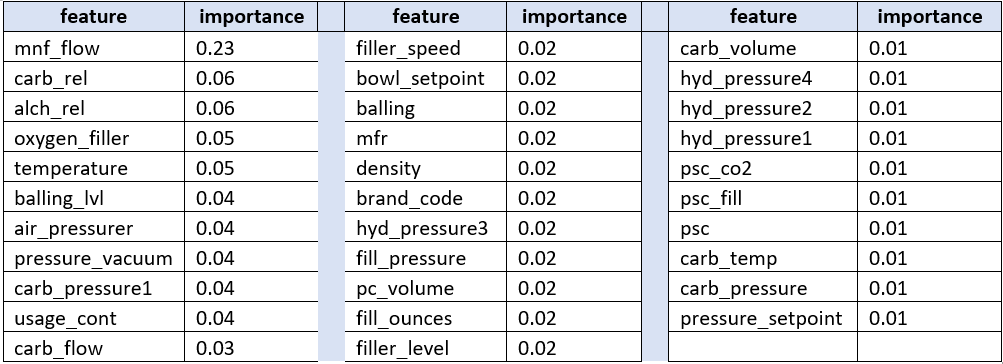
was selected because Python’s sklearn library offers the ability to easily identify features in order of importance relative to the PH that we are trying to predict. Knowing this information makes it much more intuitive to narrow down to the optimal set of features to predict PH.

The first RF model was built with all the features to show us the predictive ability of each feature. Below is a summary of the results for the 32 features…

Importance is calculated using a machine learning equation called the “gini value”. Note that all the importance values sum to 100%. For example, the table below is stating mnf\_flow contributes 23% versus the next best of carb\_rel and alch\_rel at 6% each.

|  |
| --- |
| train, test = train\_test\_split(data,test\_size=0.3) X\_train = train.drop('ph', axis=1) y\_train = train['ph']  # Create a random forest regressor rfall = RandomForestRegressor(n\_estimators=10000, random\_state=0, n\_jobs=-1)  # Train the regressor rfall.fit(X\_train, y\_train) # Print the name and gini importance of each feature rfall\_gini\_values\_df = pd.DataFrame(columns=['feature','importance']) for feature in zip(list(X\_train.columns), rfall.feature\_importances\_):  rfall\_gini\_values\_df = rfall\_gini\_values\_df.append({'feature': feature[0], 'importance': round(feature[1],2)},  ignore\_index=True) rfall\_gini\_values\_df = rfall\_gini\_values\_df.sort\_values('importance', ascending=False) rfall\_gini\_values\_df.to\_excel('data/' + 'all\_features\_ranked\_by\_gini\_values.xlsx', index=False) rfall\_gini\_values\_df |

**Summary of each feature’s importance:**

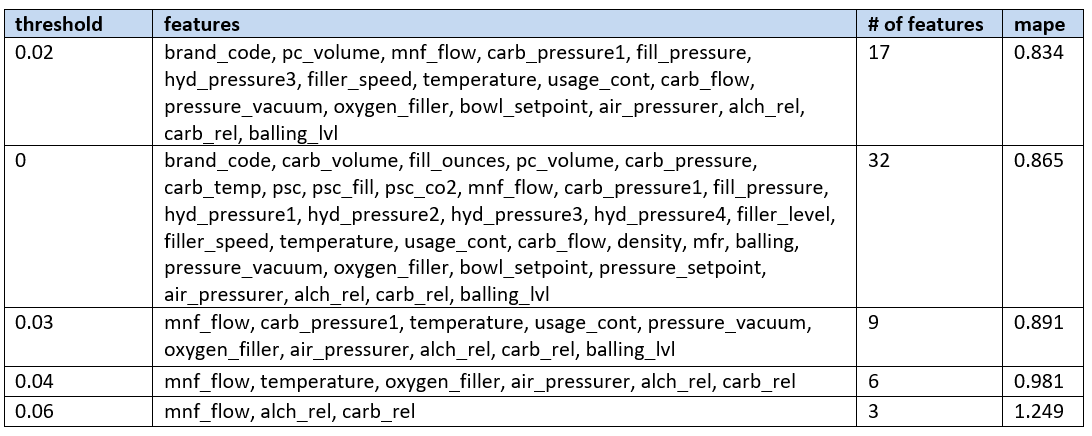


# Model Selection

Per this project’s requirements, MAPE was used to identify the best performing model and Random Forest has the lowest MAPE. With Random Forest, we tried out 6 different sets of features summarized in the table below…

|  |
| --- |
| best\_model\_candidates\_df = pd.DataFrame(columns=['threshold', 'features', 'mape'])  #Create a set of the gini values rounded to two decimal places thresholds\_set = sorted(set(rfall\_gini\_values\_df.importance.values)) thresholds\_set = thresholds\_set[:-1] thresholds\_set for g in thresholds\_set:  rfselector = SelectFromModel(rfall, threshold=g)  rfselector.fit(X\_train, y\_train)    # Transform the data to create a new dataset containing only the most important features  X\_important\_train = rfselector.transform(X\_train)  X\_important\_test = rfselector.transform(X\_test)    # Create a new random forest regressor for the most important features  rfbest = RandomForestRegressor(n\_estimators=10000, random\_state=0, n\_jobs=-1)  rfbest.fit(X\_important\_train, y\_train)  y\_rfbest = rfbest.predict(X\_important\_test)  features\_list = []  for feature\_list\_index in rfselector.get\_support(indices=True):  features\_list.append(X\_train.columns[feature\_list\_index])   if len(features\_list) == 0:  features\_list.append('no features')    #Calculate MAPE  models\_mape = round(mape(y\_test, y\_rfbest),3)   best\_model\_candidates\_df = best\_model\_candidates\_df.append({'threshold':g, 'features':features\_list, 'mape':models\_mape}, ignore\_index=True) best\_model\_candidates\_df = best\_model\_candidates\_df.sort\_values('mape') best\_model\_candidates\_df.to\_excel('data/' + 'best\_model\_candidates\_output\_ken.xlsx', index=False) best\_model\_candidates\_df |

**Summary of RF Models...**



As noted previously in the Executive Summary there is less than a half percent difference between the best model with 17 features and the “worst” (still a great MAPE), model of only 3 features.

Given this result, serious consideration should be given to whether the half percent better MAPE is worth the additional complexity of a 17 feature model versus a 3 feature model in Production.

# Prediction

|  |
| --- |
| y\_final\_predictions = rfbest.predict(X\_important\_final\_predictions) final\_predictions\_df['ph'] = y\_final\_predictions |

Our prediction results can be viewed in the spreadsheet that accompanies this write-up. Here is a link to the spreadsheet containing our predictions: link to spreadsheet.

# Conclusion

After exploration of the data we found from the beginning that the data would be difficult to build a traditional predictive model that would be able to predict the target variable with meaningful accuracy given the dispersion in our EDA plots. Right away a tree based approach yielded better results. Often when creating ML models that don’t display linear relationships between features and target will yield poor accuracy. After extensive testing we found that a random forest model had the best results. Random forest models make decisions based on information gain to make predictions. Think of the game where you guess attributes about a person in a cohort to narrow down search to one person. A tree based model employs this tactic to make predictions based on the attributes (features) of a dataset. To improve our predictions we would first look to do additional feature engineering. Improving the quality of data into the model is always the best answer to improve overall performance. It’s also important to note that it’s not a particularly large data set. Ignored to create a model that can be used in production, we would want to use a larger dataset that has a larger dispersion of feature importance to avoid any bias that may occur in our sample.