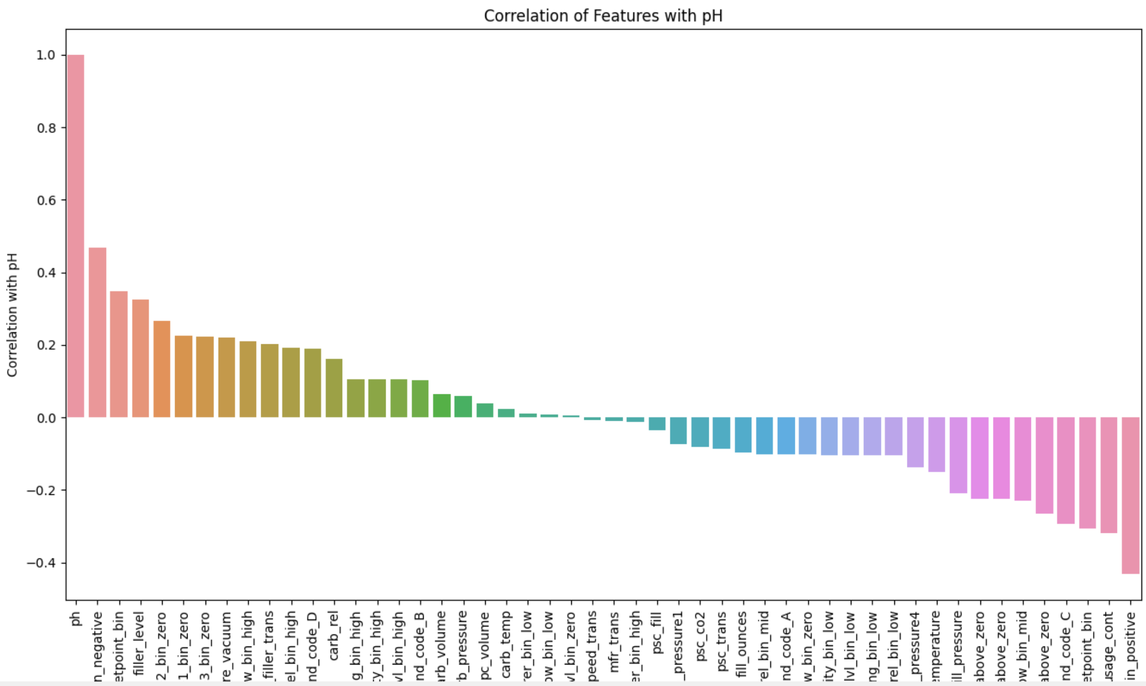
Taha and Chen

#variableSelecting #modelBuilding #modelEvaluating #ModelSelecting #ModelPredciting

```python
#Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.cross_decomposition import PLSRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.pipeline import make_pipeline
```

```python
#in this section we are doing variable selection because we want to reduce the number of predictors not revelant to the variable ph
#this will improve model prediction performance for predicting ph
#make it a little easier to intrepret the model
#reduce the cost of computation because this is a business for ABC beverages
#in the print statements/graphs there are numbers and indicators that state which predictors are not strong correlators to ph and we are opting not to use them in our models

#Loading the Test Data for variable selection, model building, model and mode predicting
testingDataPath = r"C:\Users\Rongc\Desktop\Project2\TestingDataProcessed.csv"
testingData = pd.read_csv(testingDataPath)

#convert categorical variables to numeric using one-hot encoding
testingDataNumeric = pd.get_dummies(testingData)

#PH is the target variable our boss wants for ABC beverages
targetVariable = testingDataNumeric['ph']
features = testingDataNumeric.drop('ph', axis=1)

#calculating the correlation matrix for the numeric dataset
correlationMatrix = testingDataNumeric.corr()
phCorrelations = correlationMatrix['ph'].sort_values(ascending=False)

#selecting variables that are strong correlators to ph
#we used 0.5 and 0.3 because they are generally common for stats
correlationThresholdStrong = 0.5 # means theres a strong correlation
correlationThresholdModerate = 0.3 # means there is a moderate correlation
selectedFeaturesStrong = phCorrelations[(phCorrelations >= correlationThresholdStrong) | (phCorrelations <= -correlationThresholdStrong)].index
selectedFeaturesModerate = phCorrelations[(phCorrelations >= correlationThresholdModerate) & (phCorrelations < correlationThresholdStrong) |
                                          (phCorrelations <= -correlationThresholdModerate) & (phCorrelations > -correlationThresholdStrong)].index
selectedFeaturesWeak = phCorrelations[abs(phCorrelations) < correlationThresholdModerate].index

#combining strong and moderate together
selectedFeatures = selectedFeaturesStrong.union(selectedFeaturesModerate).drop('ph')

#Prints of the actual value and the corresponding correlation to ph
print("Features with strong and moderate correlation to pH:")
for feature in selectedFeatures:
    correlation_value = phCorrelations[feature]
    correlation_category = "Strong" if abs(correlation_value) >= correlationThresholdStrong else "Moderate"
    print(f"{feature} ({correlation_category} Correlation): {correlation_value}")

print("\nFeatures with weak or no correlation to pH:")
for feature in selectedFeaturesWeak:
    if feature != 'ph':
        print(f"{feature}: {phCorrelations[feature]}")

#graph to show a visual of the correlation of variables correlating to ph
plt.figure(figsize=(15, 8))
sns.barplot(x=phCorrelations.index, y=phCorrelations.values)
plt.xticks(rotation=90)
plt.xlabel('Features')
plt.ylabel('Correlation with pH')
plt.title('Correlation of Features with pH')
plt.show()
```

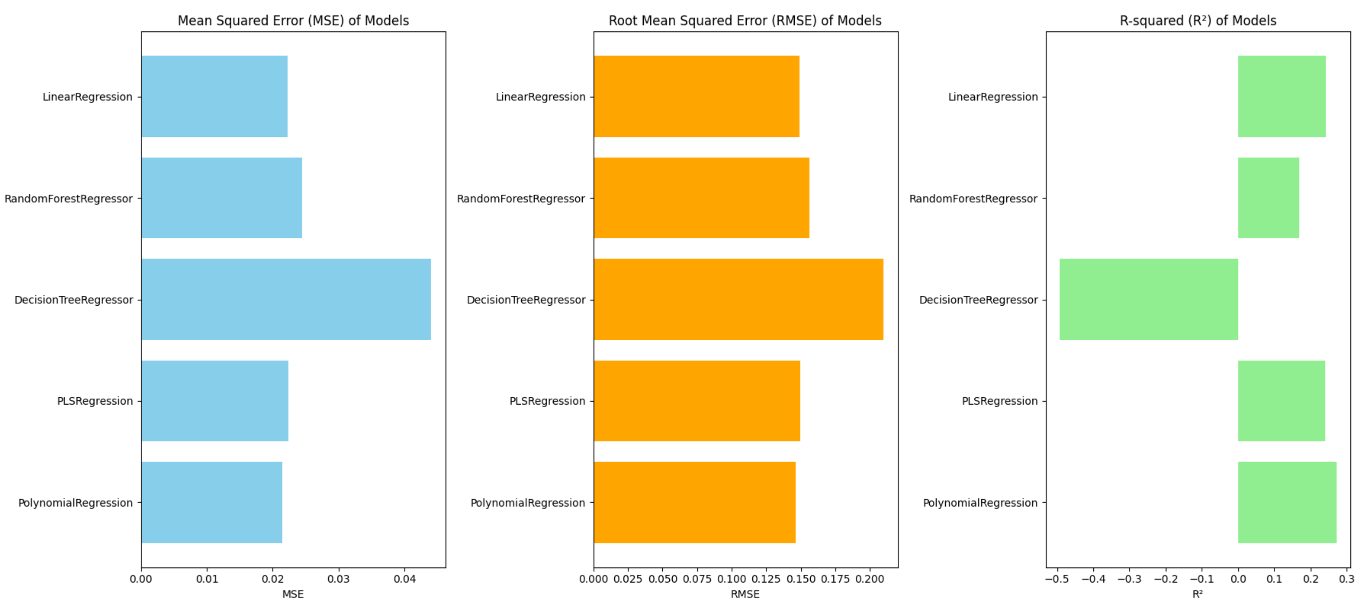



```
Features with strong and moderate correlation to pH:
bowl_setpoint_bin (Moderate Correlation): 0.34829243501643675
filler_level (Moderate Correlation): 0.32430482945545946
mnf_flow_bin_negative (Moderate Correlation): 0.4685449173252741
mnf_flow_bin_positive (Moderate Correlation): -0.43157431007045766
pressure_setpoint_bin (Moderate Correlation): -0.3070616792918806
usage_cont (Moderate Correlation): -0.3181072262308761
```

```
Features with weak or no correlation to pH:
hyd_pressure2_bin_zero: 0.26620694288279967
hyd_pressure1_bin_zero: 0.22494341678000382
hyd_pressure3_bin_zero: 0.22377493976672705
pressure_vacuum: 0.2205096730713491
carb_flow_bin_high: 0.21088669708682595
oxygen_filler_trans: 0.20153676544522764
alch_rel_bin_high: 0.19227532498453223
brand_code_D: 0.18923596866970802
carb_rel: 0.1613853106502643
balling_bin_high: 0.1048831949552594
density_bin_high: 0.10431127780714758
balling_lvl_bin_high: 0.10417738787211916
brand_code_B: 0.10292672422256868
carb_volume: 0.06501150608141244
carb_pressure: 0.05994004541575252
pc_volume: 0.038834777916828656
carb_temp: 0.022564168837281105
air_pressurer_bin_low: 0.011651116286033499
carb_flow_bin_low: 0.007180262610724045
balling_lvl_bin_zero: 0.005263160077082794
filler_speed_trans: -0.006986984315991737
mfr_trans: -0.01093539758478368
air_pressurer_bin_high: -0.011651116286033504
psc_fill: -0.03427413239093513
carb_pressure1: -0.07331059235736752
psc_co2: -0.08190252156363692
psc_trans: -0.08723983278037595
fill_ounces: -0.096366747216344
alch_rel_bin_mid: -0.10088920799204425
brand_code_A: -0.10253060430975168
mnf_flow_bin_zero: -0.10255776356389108
density_bin_low: -0.1043112778071476
balling_lvl_bin_low: -0.10456875885313943
balling_bin_low: -0.10488319495525913
alch_rel_bin_low: -0.10527376868612105
hyd_pressure4: -0.13862235628512254
temperature: -0.1496250586303669
fill_pressure: -0.21028379530359573
hyd_pressure3_bin_above_zero: -0.22377493976672694
hyd_pressure1_bin_above_zero: -0.22494341678000387
carb_flow_bin_mid: -0.23041685408114532
hyd_pressure2_bin_above_zero: -0.2662069428827997
brand_code_C: -0.29337174939846905
```

```python
#model Building
#in this section we are building multiple different models that comes with its own set of assumptions about the data
#it is for performance because different models can perform differently on various datasets. Some models might work well with a particular type of data distribution
#we chose linear, random forest, decision tree, Partial least squared, and polynomial
#because linear serves as a good baseline for us as it's the simplest most easy to interpret
#poly for the second easiest and allows non linear relationships
#random forest to capture relationships that are compelex in the data
#tree decision for non linear relationships
#pls for multicollinearity
selectedFeaturesData = testingDataNumeric[selectedFeatures]

#splitting the dataset into training and testing sets
XTrain, XTest, YTrain, YTest = train_test_split(selectedFeaturesData, targetVariable, test_size=0.2, random_state=42)

#models
models = {
    'LinearRegression': LinearRegression(),
    'RandomForestRegressor': RandomForestRegressor(n_estimators=100, random_state=42),
    'DecisionTreeRegressor': DecisionTreeRegressor(random_state=42),
    'PLSRegression': PLSRegression(n_components=2),
    'PolynomialRegression': make_pipeline(PolynomialFeatures(degree=2), LinearRegression())
}

#training and evaluation data
modelStats = {}
for modelName, model in models.items():
    model.fit(XTrain, YTrain)
    predictions = model.predict(XTest)
    mse = mean_squared_error(YTest, predictions)
    rmse = np.sqrt(mse)  # Calculating RMSE
    r2 = r2_score(YTest, predictions)
    modelStats[modelName] = {'MSE': mse, 'RMSE': rmse, 'R2': r2}

#model performance
for modelName, stats in modelStats.items():
    print(f"{modelName} - MSE: {stats['MSE']:.3f}, RMSE: {stats['RMSE']:.3f}, R²: {stats['R2']:.3f}")


modelNames = list(modelStats.keys())
y_pos = np.arange(len(modelNames))
mseValues = [stats['MSE'] for stats in modelStats.values()]
rmseValues = [stats['RMSE'] for stats in modelStats.values()]
r2Values = [stats['R2'] for stats in modelStats.values()]
```

```python
modelNames = list(modelStats.keys())
y_pos = np.arange(len(modelNames))
mseValues = [stats['MSE'] for stats in modelStats.values()]
rmseValues = [stats['RMSE'] for stats in modelStats.values()]
r2Values = [stats['R2'] for stats in modelStats.values()]

fig, ax = plt.subplots(1, 3, figsize=(18, 8))

ax[0].barh(y_pos, mseValues, align='center', color='skyblue')
ax[0].set_yticks(y_pos)
ax[0].set_yticklabels(modelNames)
ax[0].invert_yaxis()
ax[0].set_xlabel('MSE')
ax[0].set_title('Mean Squared Error (MSE) of Models')

ax[1].barh(y_pos, rmseValues, align='center', color='orange')
ax[1].set_yticks(y_pos)
ax[1].set_yticklabels(modelNames)
ax[1].invert_yaxis()
ax[1].set_xlabel('RMSE')
ax[1].set_title('Root Mean Squared Error (RMSE) of Models')

ax[2].barh(y_pos, r2Values, align='center', color='lightgreen')
ax[2].set_yticks(y_pos)
ax[2].set_yticklabels(modelNames)
ax[2].invert_yaxis()
ax[2].set_xlabel('R²')
ax[2].set_title('R-squared (R²) of Models')

plt.tight_layout()
plt.show()
```

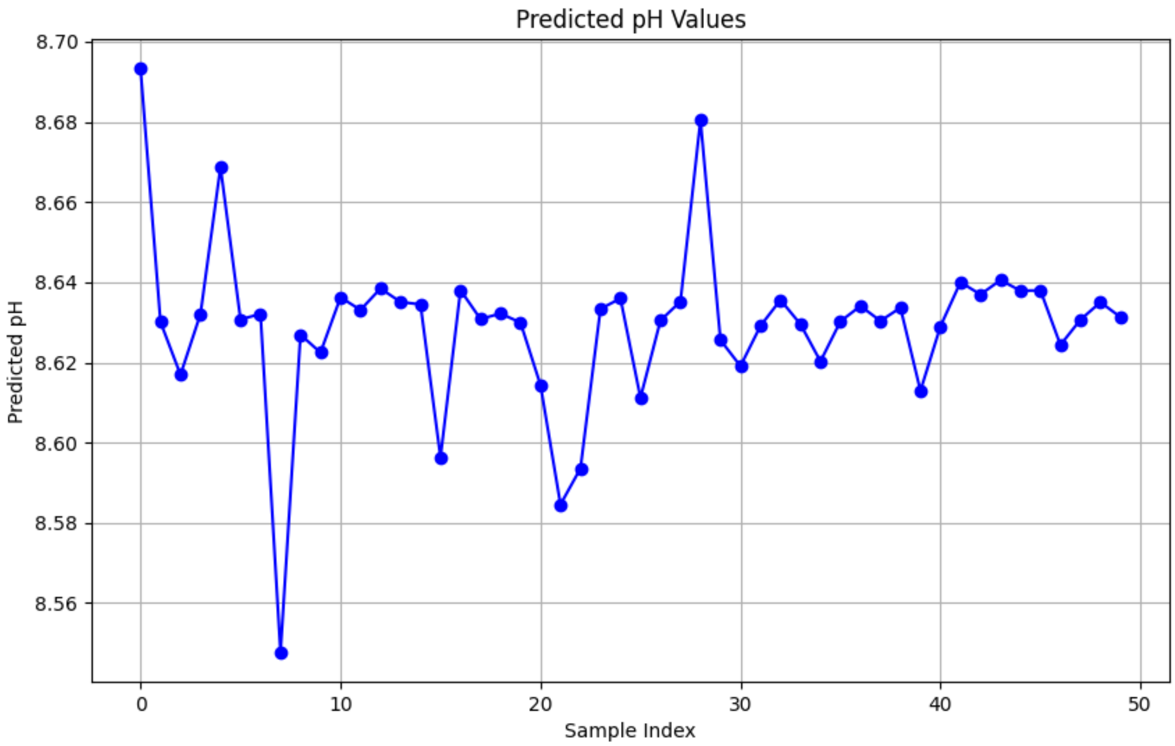



```
LinearRegression - MSE: 0.022, RMSE: 0.149, R²: 0.244
RandomForestRegressor - MSE: 0.025, RMSE: 0.157, R²: 0.170
DecisionTreeRegressor - MSE: 0.044, RMSE: 0.210, R²: -0.493
PLSRegression - MSE: 0.022, RMSE: 0.150, R²: 0.241
PolynomialRegression - MSE: 0.021, RMSE: 0.147, R²: 0.272
```

```python
#selecting the model
#taking in the consideration for MSE, RMSE, and R2 out of all the models
#and comparing them to find the optimal model to predict ph
#polynomal model out performed the other models in all 3 key metrics together.
bestModelName = None
bestModelStats = {'MSE': np.inf, 'RMSE': np.inf, 'R2': -np.inf}

for modelName, stats in modelStats.items():
    if stats['MSE'] < bestModelStats['MSE'] and stats['RMSE'] < bestModelStats['RMSE'] and stats['R2'] > bestModelStats['R2']:
        bestModelName = modelName
        bestModelStats = stats

# Print the best model's performance
print(f"Best Performing Model: {bestModelName}")
print(f"MSE: {bestModelStats['MSE']:.3f}, RMSE: {bestModelStats['RMSE']:.3f}, R²: {bestModelStats['R2']:.3f}")
```

```
Best Performing Model: PolynomialRegression
MSE: 0.021, RMSE: 0.147, R²: 0.272
```

```python
#lastly we use the evaluation data with the best performing model for the prediction of PH
evaluationDataPath = r"C:\Users\Rongc\Desktop\Project2\EvaluationData.csv"
evaluationData = pd.read_csv(evaluationDataPath)
evaluationDataNumeric = pd.get_dummies(evaluationData)
evaluationDataFeatures = evaluationDataNumeric[selectedFeatures]

bestModel = models[bestModelName]
predictedPH = bestModel.predict(evaluationDataFeatures)

evaluationData['predictedPH'] = predictedPH
print(evaluationData[['predictedPH']])

#graph for ph
plt.figure(figsize=(10, 6))
plt.plot(evaluationData['predictedPH'].head(50), marker='o', linestyle='-', color='blue')
plt.title('Predicted pH Values')
plt.xlabel('Sample Index')
plt.ylabel('Predicted pH')
plt.grid(True)
plt.show()
```



Predicted pH Values
Predicted pH
Sample Index
8.70
8.68
8.66
8.64
8.62
8.60
8.58
8.56
0
10
20
30
40
50