Baker Hughes Hackathon

Predictive Modeling (Team 2)

Isaura Ramírez Salazar
José David Romo López
Juan Diego Sanchez Díaz

Goal

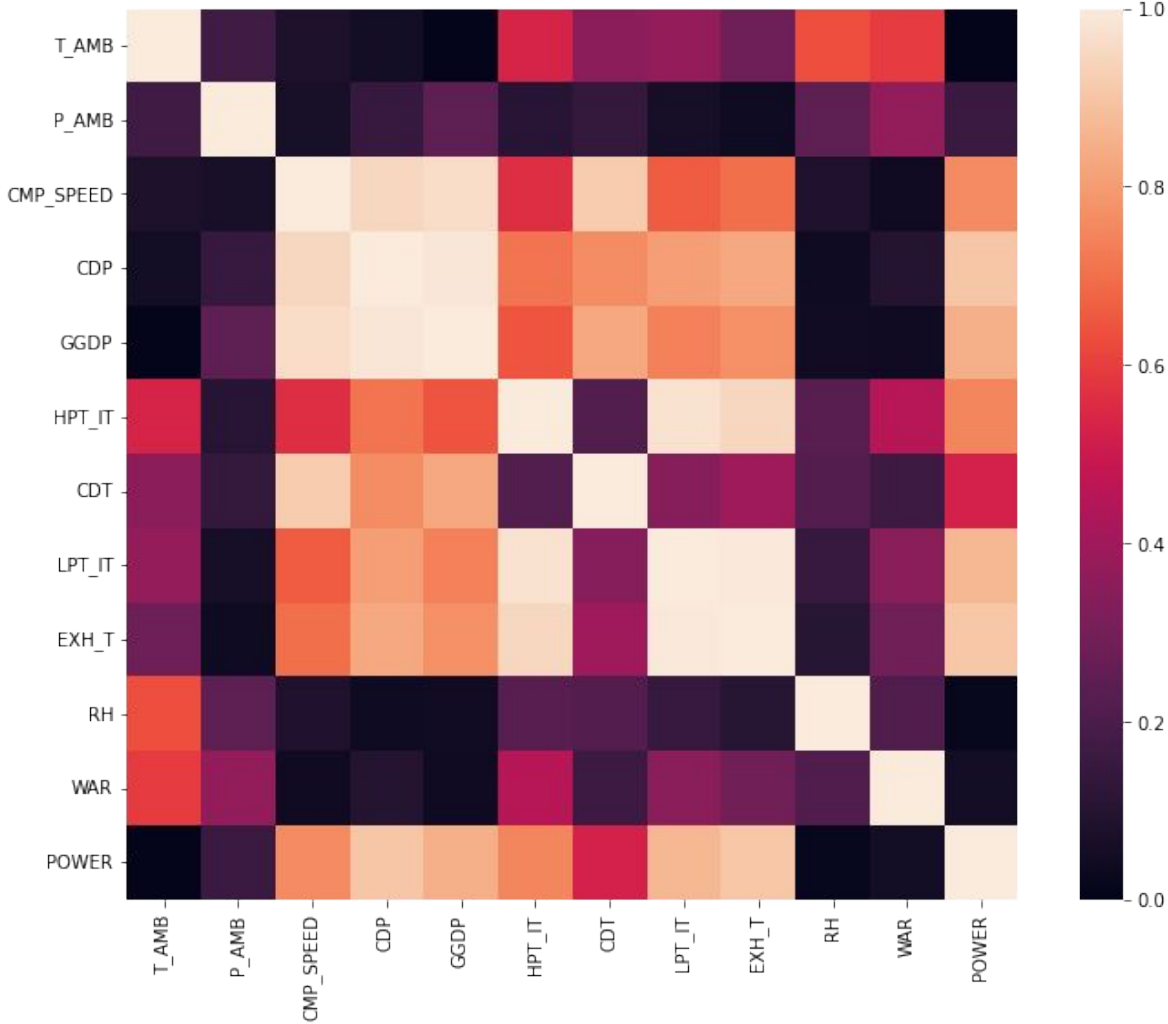
Obtain a predictor using turbine data

Data Transformation

Data Visualization

(all training data)

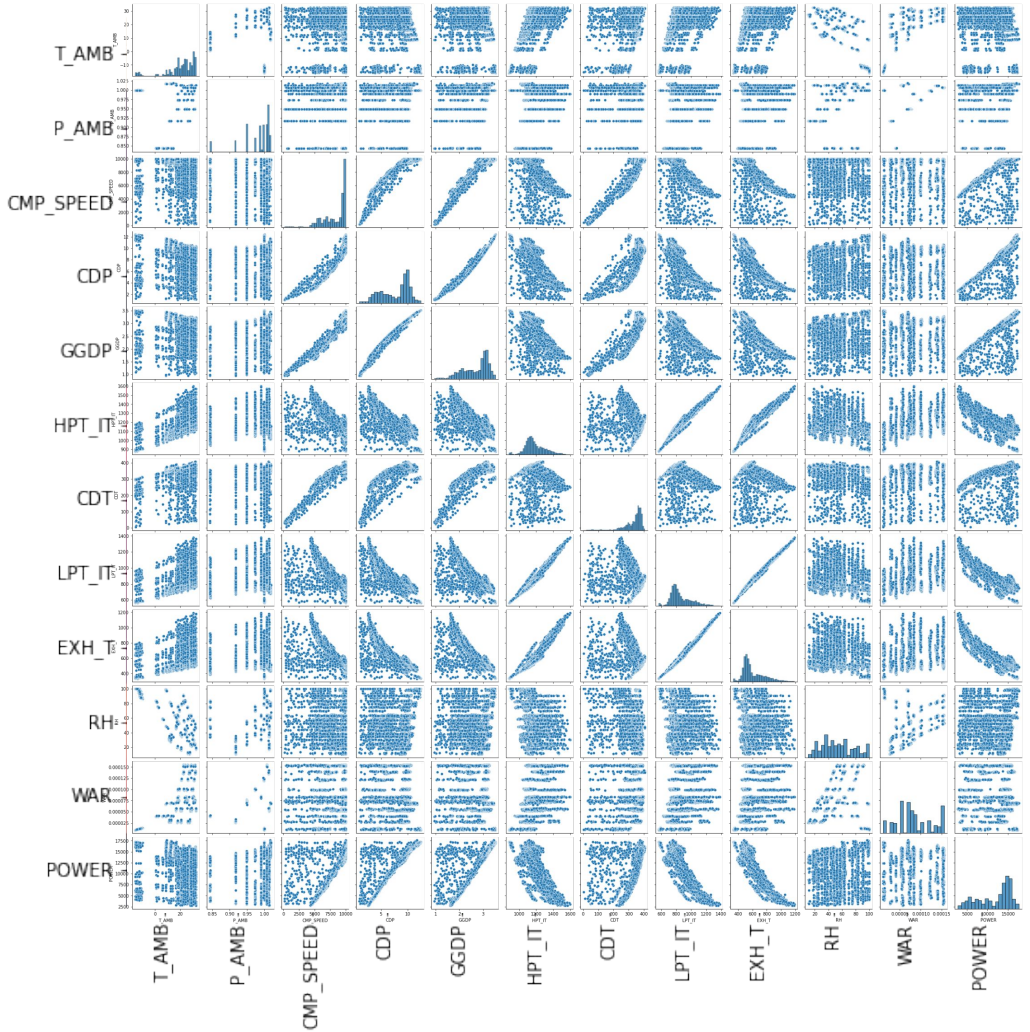
abs(CorrelationCoeff)
The clearer the better



Data Visualization

(all training data)

abs(CorrelationCoeff)
The clearer the better



For us the interesting variables are in order:

```
POWER        1.000000
EXH_T        0.903272
CDP          0.898743
LPT_IT       0.863357
GGDP         0.846911
CMP_SPEED    0.758151
HPT_IT       0.746706
CDT          0.525526
P_AMB        0.155483
WAR          0.049835
RH           0.017927
T_AMB        0.004338
Name: POWER, dtype: float64
```

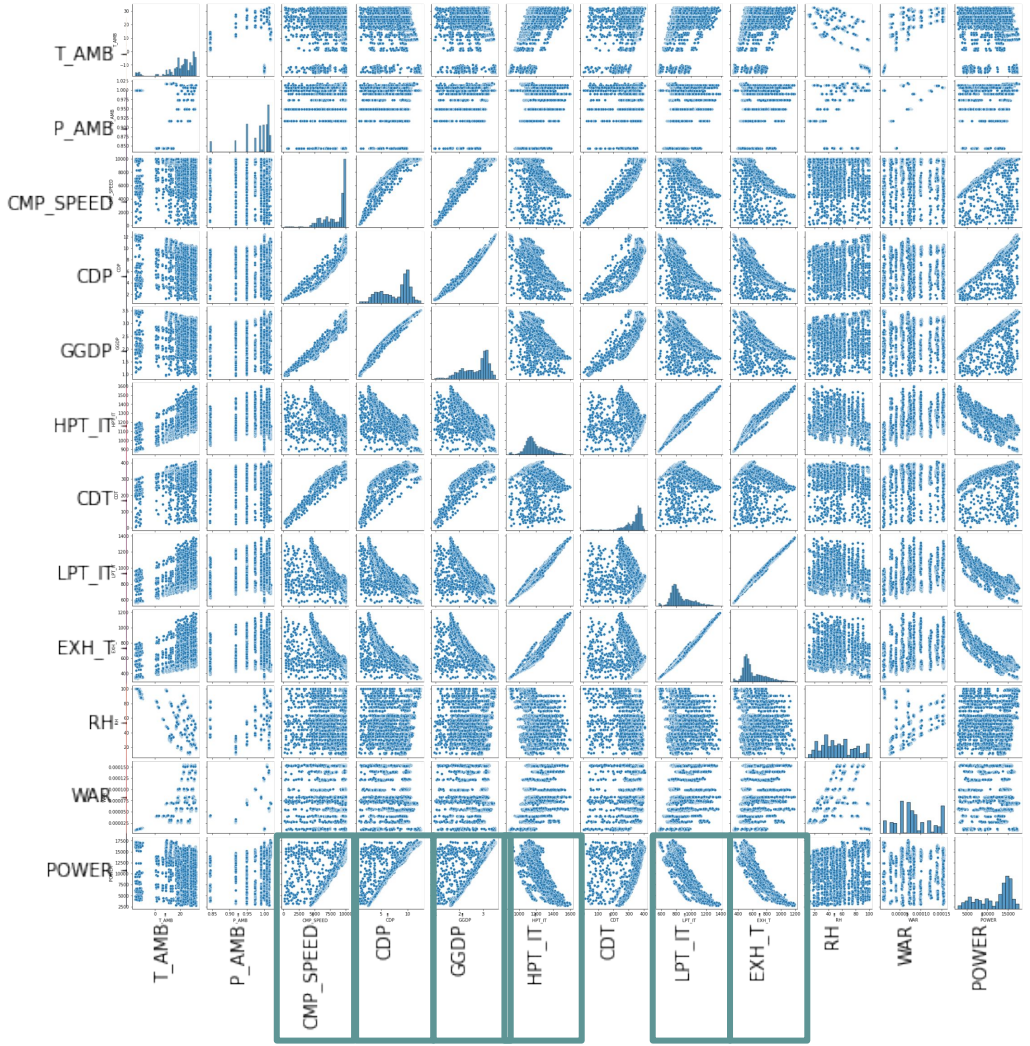
Data Visualization

(all training data)

abs(CorrelationCoeff)
The clearer the better

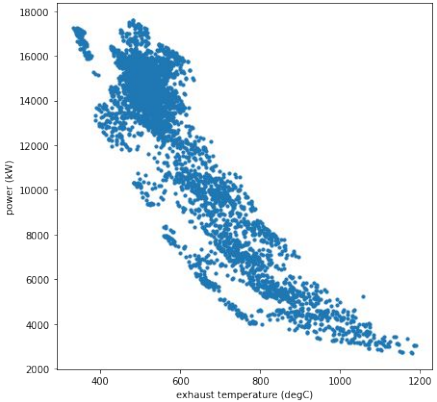
For us the interesting variables are in order:

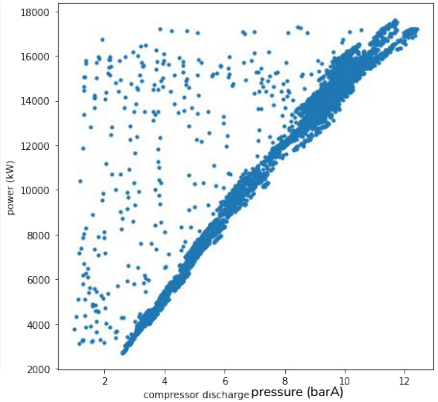
```
POWER          1.000000
EXH_T          0.903272
CDP            0.898743
LPT_IT         0.863357
GGDP           0.846911
CMP_SPEED      0.758151
HPT_IT         0.746706
CDT            0.525526
P_AMB          0.155483
WAR            0.049835
RH             0.017927
T_AMB          0.004338
Name: POWER, dtype: float64
```

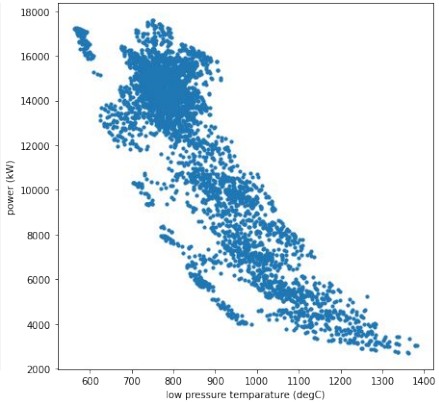


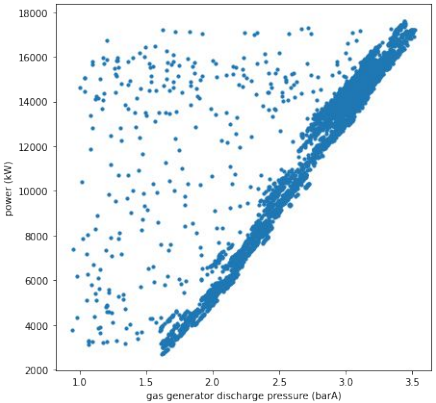
Data Visualization

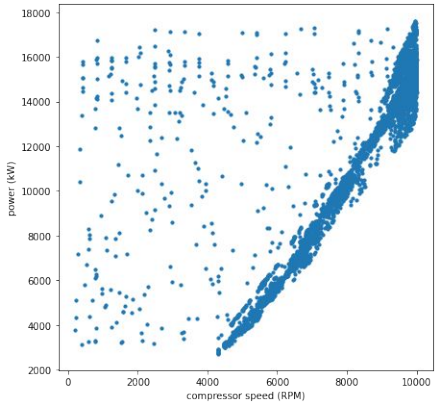
We visualize all the training data

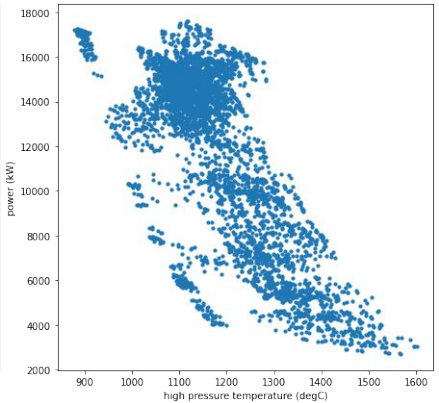










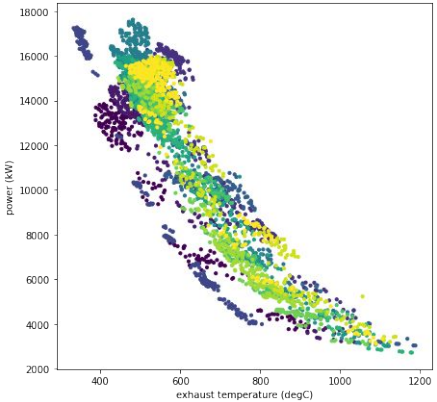


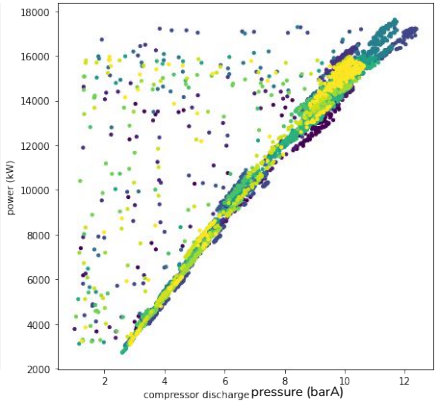

```
POWER          1.000000
EXH_T          0.903272
CDP            0.898743
LPT_IT         0.863357
GGDP           0.846911
CMP_SPEED      0.758151
HPT_IT         0.746706
CDT            0.525526
P_AMB          0.155483
WAR            0.049835
RH             0.017927
T_AMB          0.004338
Name: POWER, dtype: float64
```

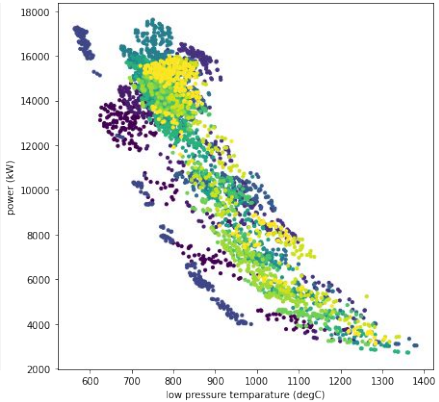
Data Visualization

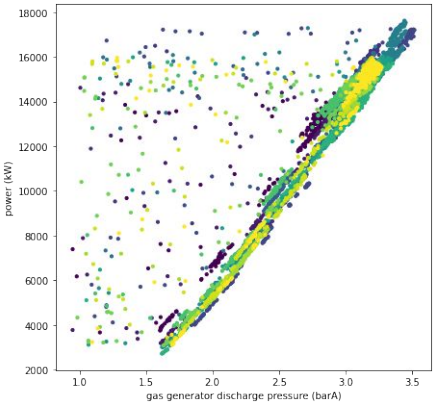
Maybe if we separate distint files. Could be a metadata depandence?

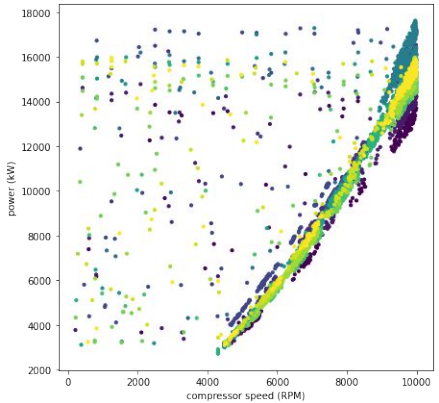
Doesn't look very important.

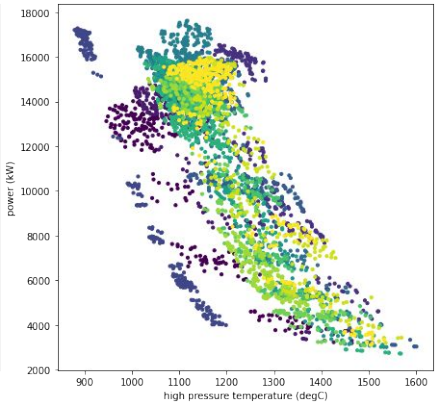












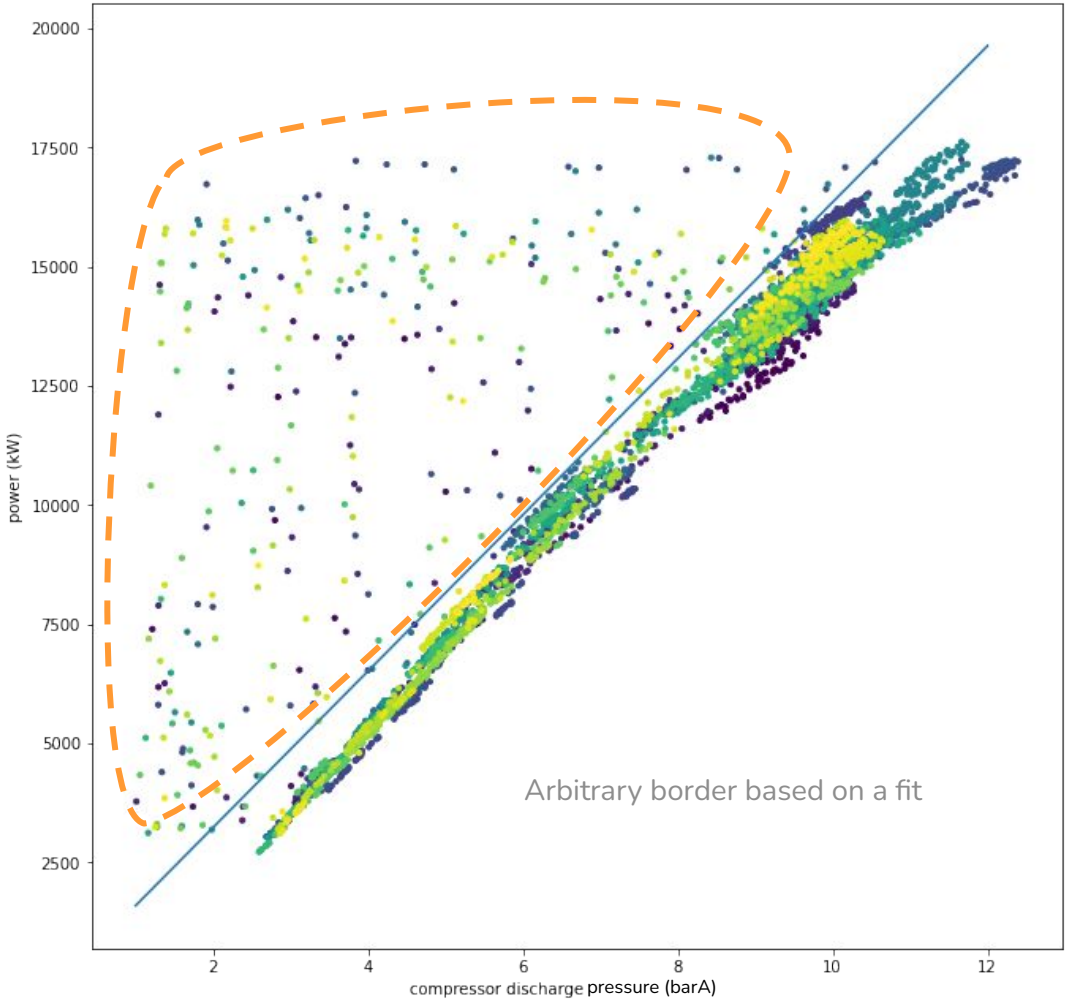

```
POWER          1.000000
EXH_T          0.903272
CDP            0.898743
LPT_IT         0.863357
GGDP           0.846911
CMP_SPEED      0.758151
HPT_IT         0.746706
CDT            0.525526
P_AMB          0.155483
WAR            0.049835
RH             0.017927
T_AMB          0.004338
Name: POWER, dtype: float64
```

Data Visualization

‘Data’ + **‘Outliers’**

Data: Below the rect
‘Outliers’: Above the rect

The dependence power vs CDP looks like a linear dependence except from the presence of some ’outliers’

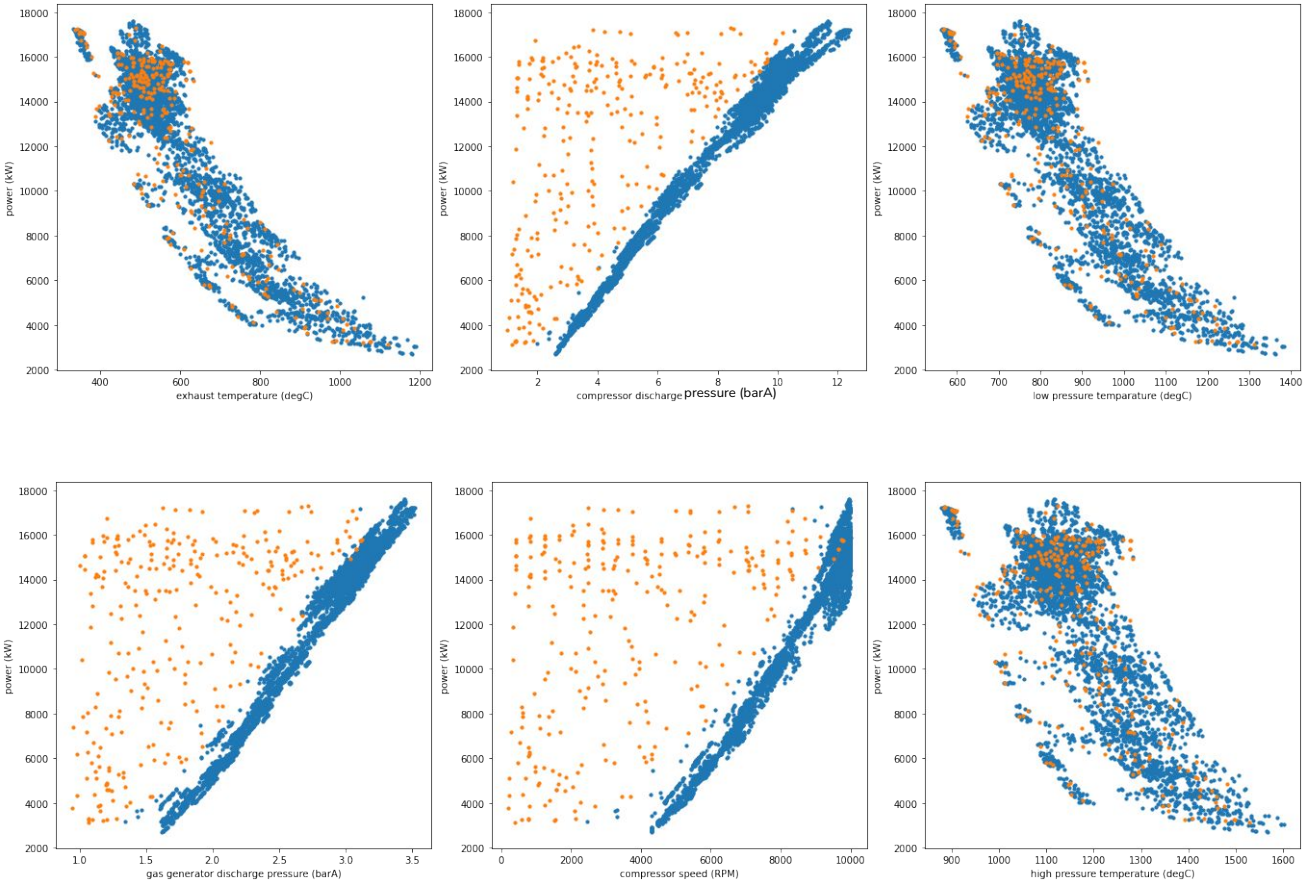


Data Visualization

'Data' + **'Outliers'**

But, are they really outliers?

- ❑ They show a consistent behaviour.
- ❑ They are not <u>rare</u> ~6% of the sample

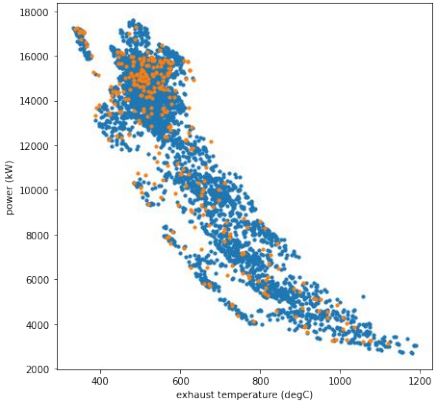


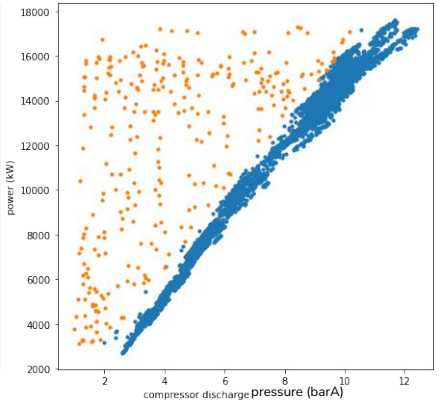
Data Visualization

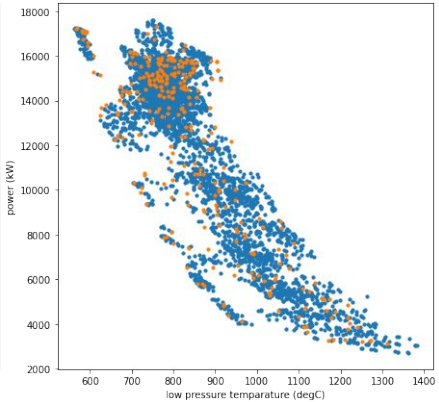
~~'Data'~~ + ~~'Outliers'~~

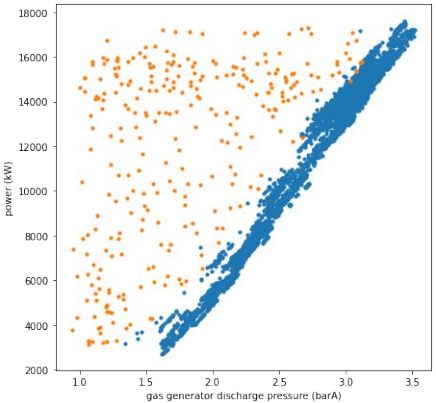
Blue type + **Orange type**

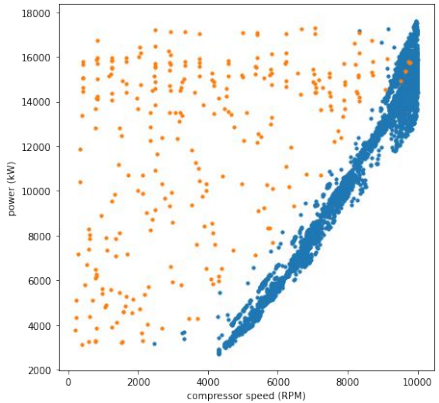
Each with a different estimator method

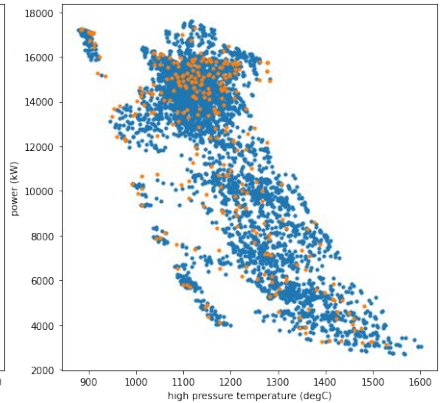








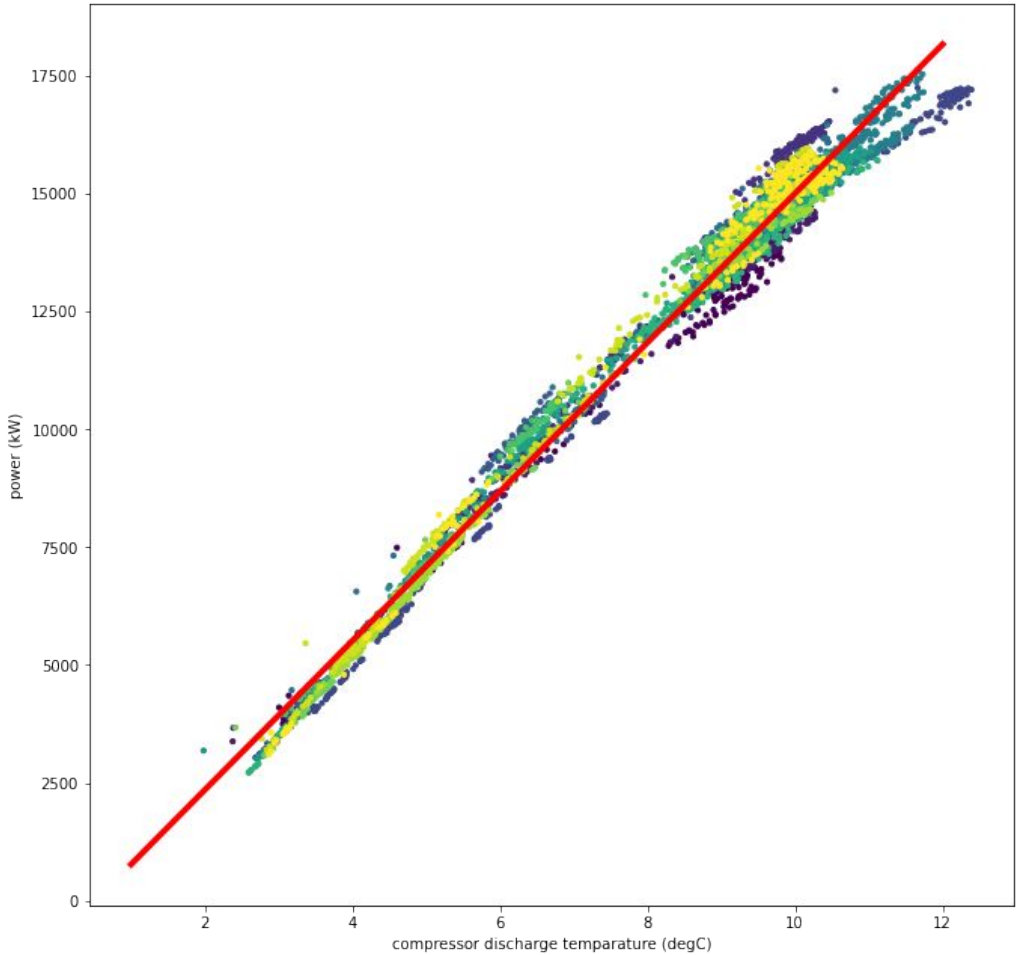




Predictors

Blue (power) predictor

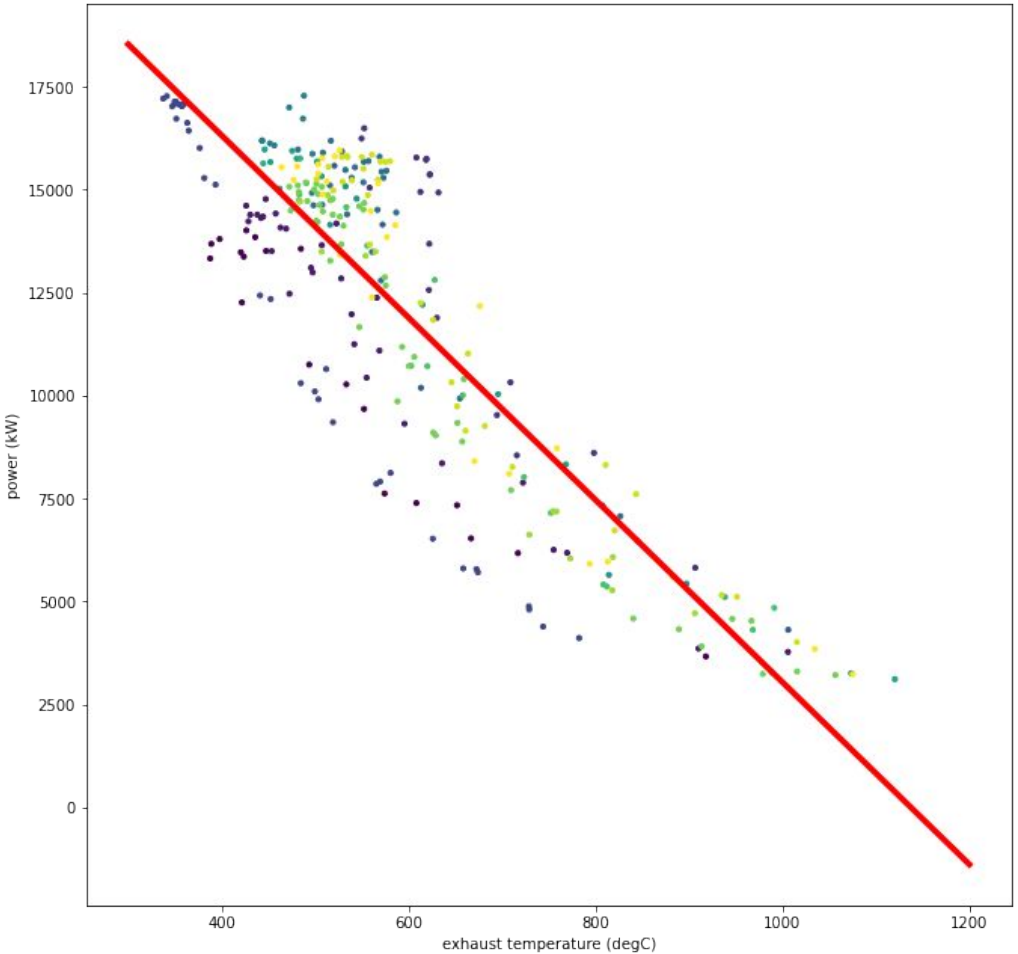
CDP vs POWER
Linear Regression



Predictors

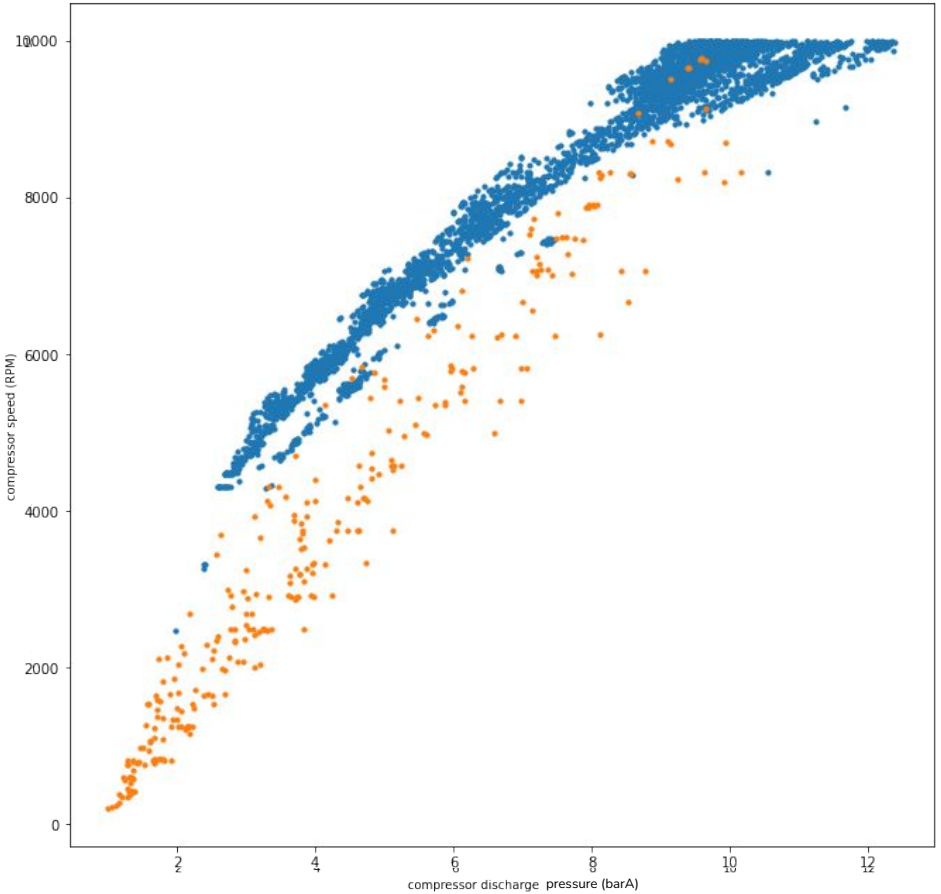
Orange (power)predictor

EXH_T vs POWER
Linear Regression



However, how can we distinguish blue vs orange points?

We could do some clustering training...



First approach

K-means

```
[ ]  x = df_01[['compressor_discharge_pressure[barA]' , 'exhaust_temperature[°C]']].to_numpy()
     y = df_01['label'].to_numpy()

     x_train , x_test , y_train , y_test = train_test_split(x , y , test_size = 0.3)
```

```
modelo_kmeans = KMeans(n_clusters = 2).fit(x_train)
centroides = modelo_kmeans.cluster_centers_
print(centroides)
```

```
[[  8.60940768 447.09607953]
 [  4.05658426 737.20382455]]
```

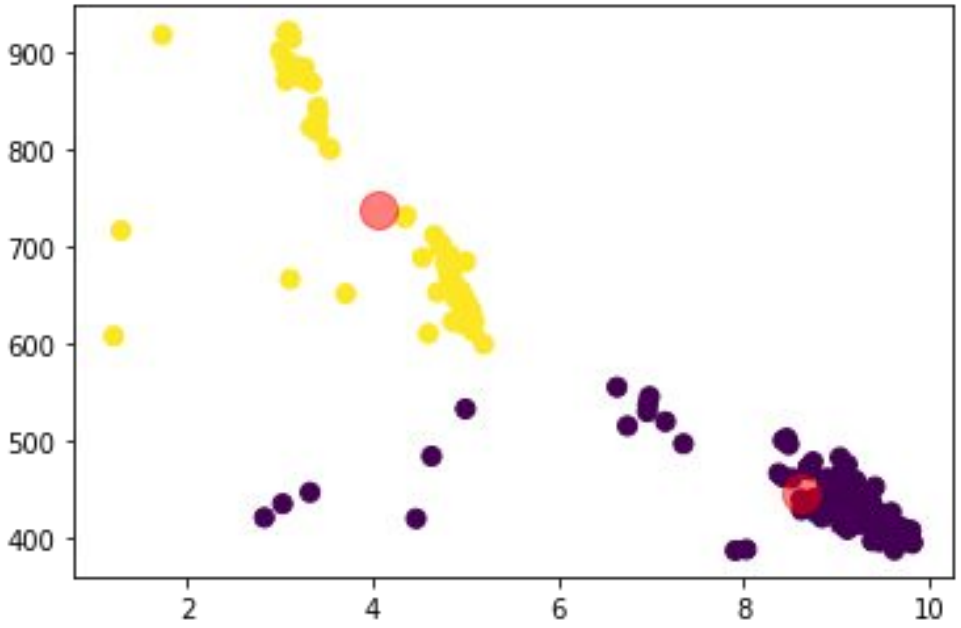
```
[ ]  y_kmeans = modelo_kmeans.predict(x_train)
```

```
[ ]  plt.scatter(x_train[:, 0], x_train[:, 1], c=y_kmeans, s=50, cmap='viridis')

     plt.scatter(centroides[:, 0], centroides[:, 1], c='red', s=200, alpha=0.5);
```

First approach

K-means

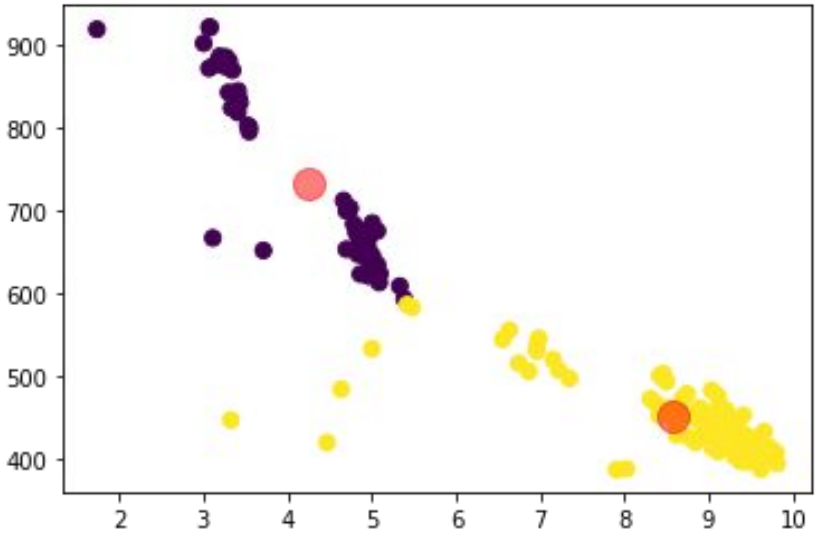


First approach

K-means

```
x = df_01[['compressor_discharge_pressure[barA]' , 'exhaust_temperature[°C]']].to_numpy()
y = df_01['power_output[kW]'].to_numpy()
```

[[4.24034601 731.48913946]
 [8.58409587 452.23910862]]

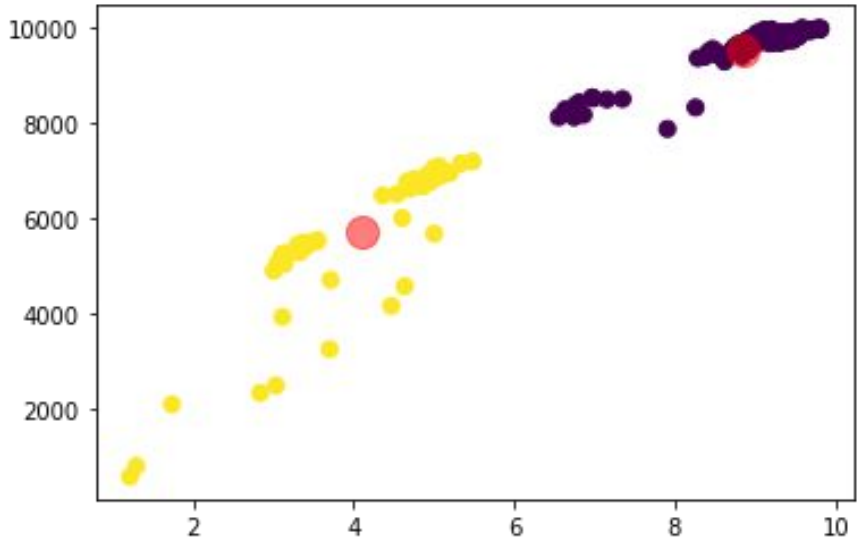


First approach

K-means

```
x = df_01[['compressor_discharge_pressure[barA]' , 'compressor_speed[RPM]']].to_numpy()
y = df_01['power_output[kW]'].to_numpy()
```

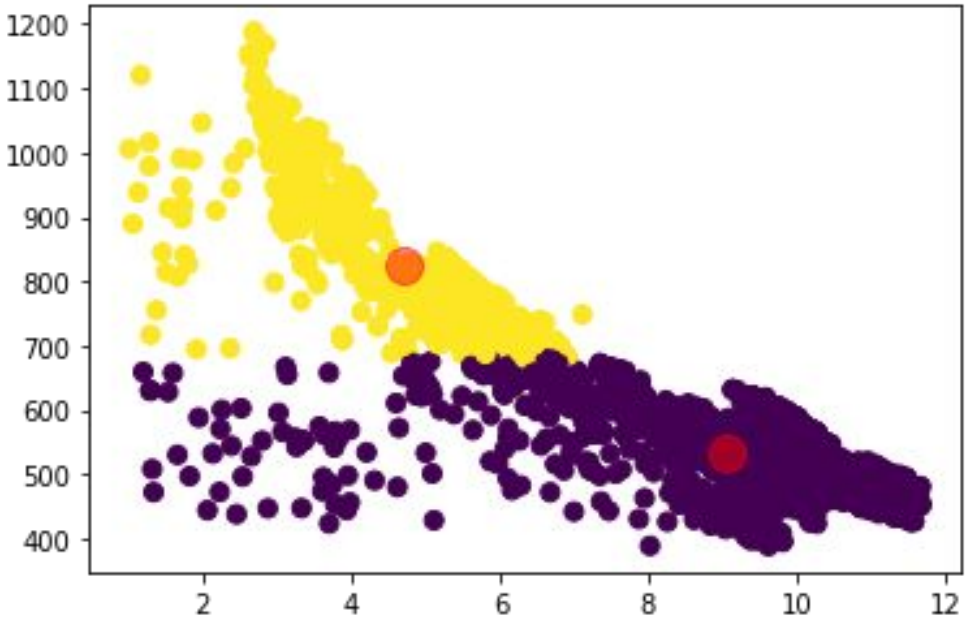
[[8.85408884e+00 9.54425045e+03]
 [4.10239771e+00 5.70094972e+03]]



K-means for complete dataset

```
x = df_01[['compressor_discharge_pressure[barA]' , 'exhaust_temperature[°C]']].to_numpy()
y = df_01['label'].to_numpy()
```

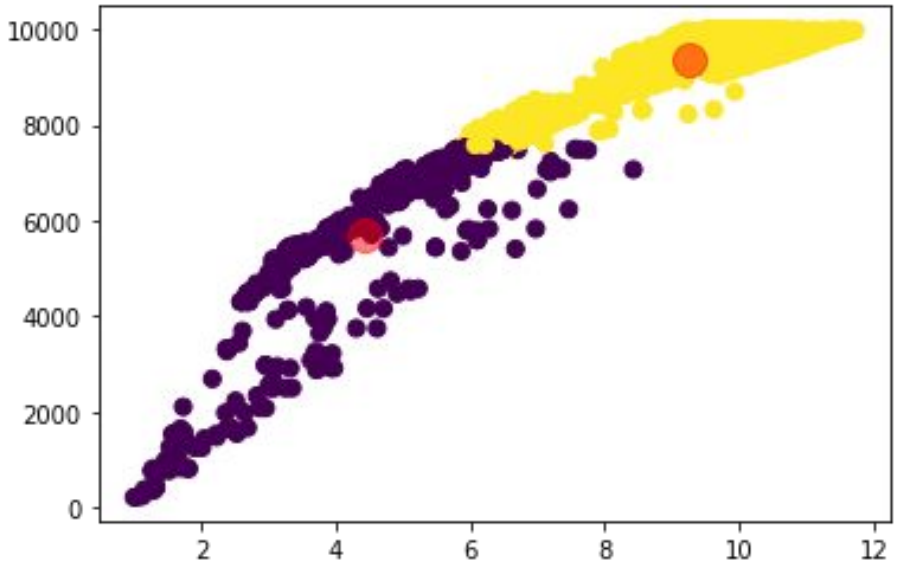
```
[[  9.06245294 531.78161744]
 [  4.69096449 825.6325629 ]]
```



K-means for complete dataset

```
x = df_01[['compressor_discharge_pressure[barA]' , 'compressor_speed[RPM]']].to_numpy()
y = df_01['power_output[kW]'].to_numpy()
```

```
[[4.44718789e+00 5.72056471e+03]
 [9.24431853e+00 9.36459111e+03]]
```



First approach

Linear Regression

x_train shape: (122, 10)
x_test shape: (82, 10)
y_train shape: (122,)
x_test shape: (82,)

Entrenamiento: MSE =1684124.1465175531
Entrenamiento: RMSE = 1297.7380885670086
Pruebas: MSE =2231558.41631598
Pruebas: RMSE = 1493.8401575523333

	valor real	prediccion	diferencia
0	13699.931524	13615.749303	84.182221
1	11832.417851	11771.131817	61.286034
2	3939.408568	4160.865158	-221.456590
3	12985.607738	12963.764149	21.843588
4	13604.550278	13486.608011	117.942267
5	4589.878313	4476.772170	113.106144
6	12845.751340	12458.006525	387.744815
7	12259.119269	11775.590208	483.529061
8	4401.739916	5089.461968	-687.722052
9	10064.378955	9759.047933	305.331022

First approach

Complete dataset

x_train shape: (2099, 10)
x_test shape: (1400, 10)
y_train shape: (2099,)
x_test shape: (1400,)

Entrenamiento: MSE =1175602.1811704403
Entrenamiento: RMSE = 1084.25189931604
Pruebas: MSE =1078603.8319800822
Pruebas: RMSE = 1038.558535654145

```
max_error(Y_test, y_test_predict)
```

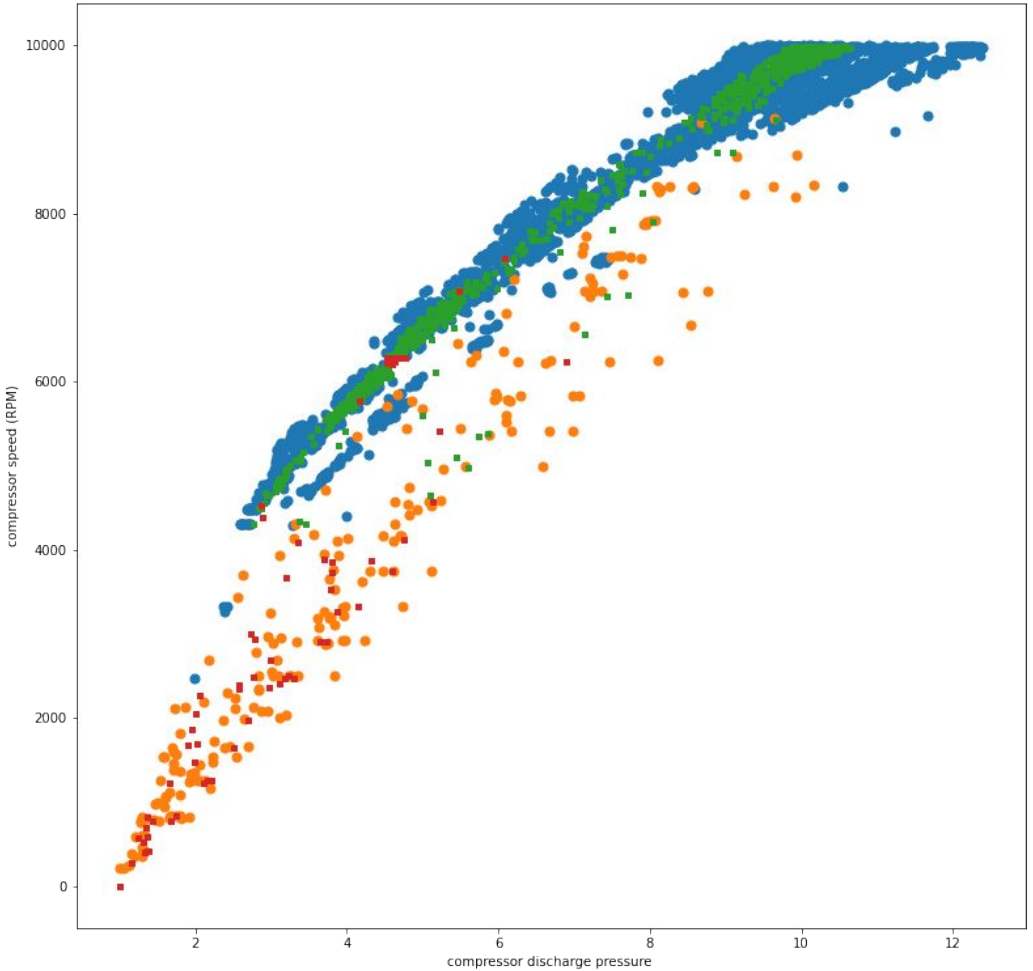
11388.37859014095

	real_value	prediction	difference
0	15045.951577	14981.763186	64.188392
1	13772.934806	13775.173671	-2.238865
2	15499.096459	15609.144670	-110.048211
3	15682.622888	15200.089142	482.533746
4	15961.827590	15928.200787	33.626803
5	14194.978578	14319.029135	-124.050557
6	13286.088486	13062.480177	223.608310
7	9926.411619	9864.001608	62.410011
8	4298.147094	4692.151744	-394.004650
9	13591.638554	13827.083420	-235.444866

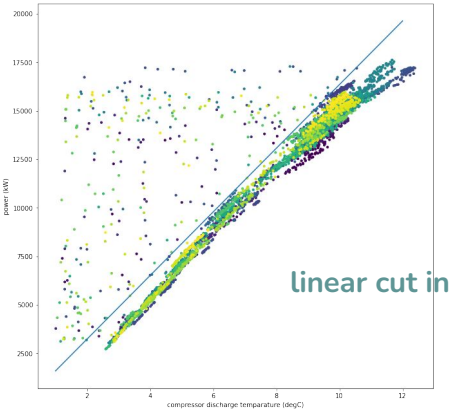
Second approach

k-nearest neighbors

- Train data (blue label from linear cut)
- Train data (orange from linear cut)
- Test data (blue label from kNearNeigh)
- Test data (orange label from kNearNeigh)



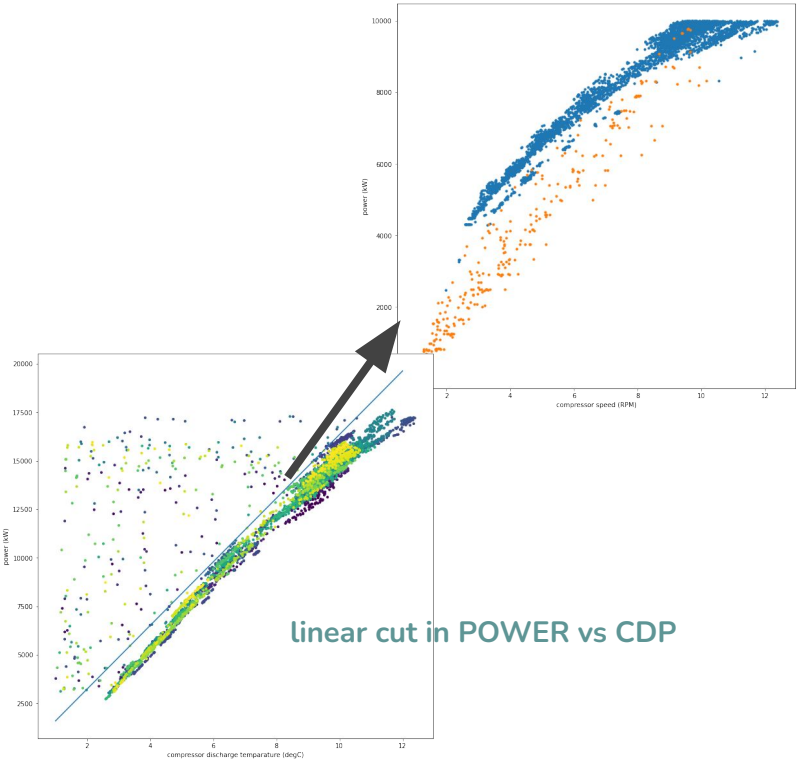
In short words...



linear cut in POWER vs CDP

In short words...

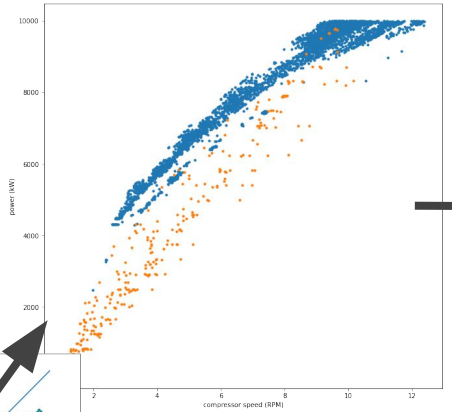
labeling data between **blue** and **orange**

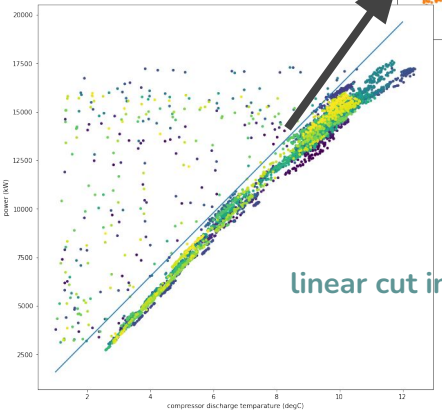


linear cut in POWER vs CDP

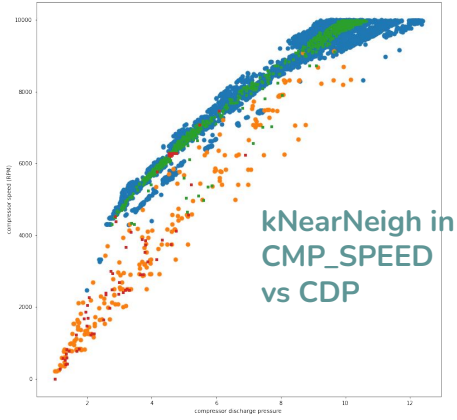
In short words...

labeling data between **blue** and **orange**





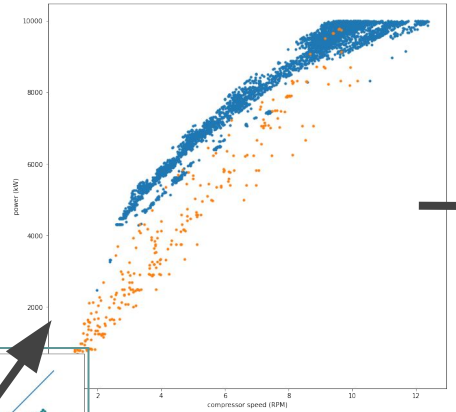
linear cut in POWER vs CDP

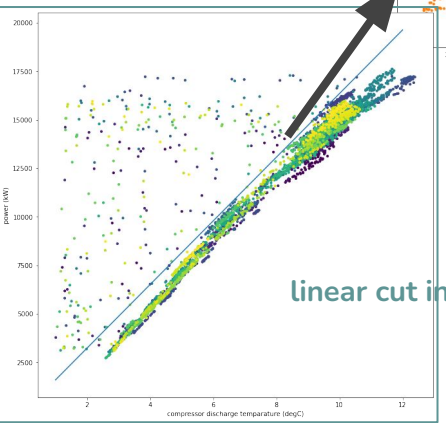


kNearNeigh in CMP_SPEED vs CDP

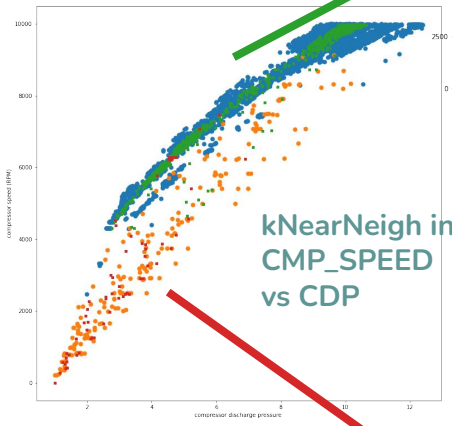
In short words...

labeling data between **blue** and **orange**

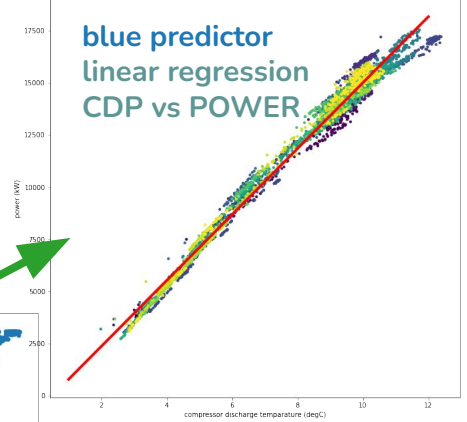




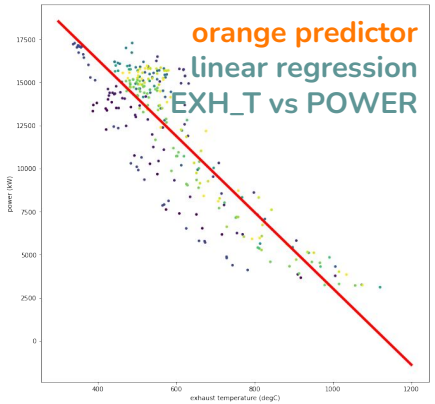
linear cut in POWER vs CDP



kNearNeigh in
CMP_SPEED
vs CDP



blue predictor
linear regression
CDP vs POWER



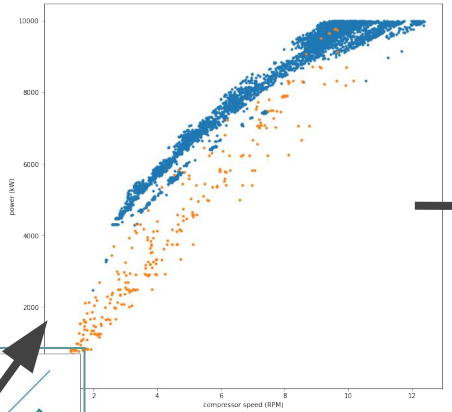
orange predictor
linear regression
EXH_T vs POWER

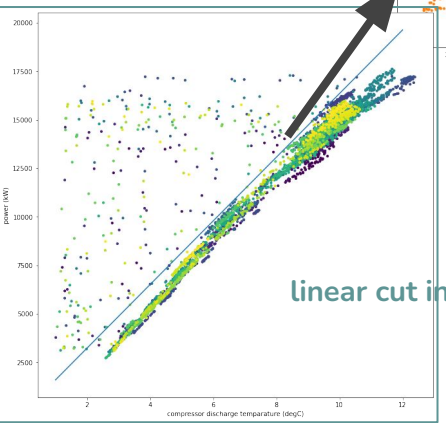
In short words...

Training

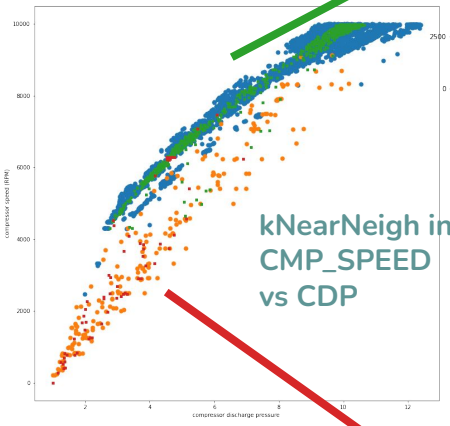
Prediction

labeling data between **blue** and **orange**

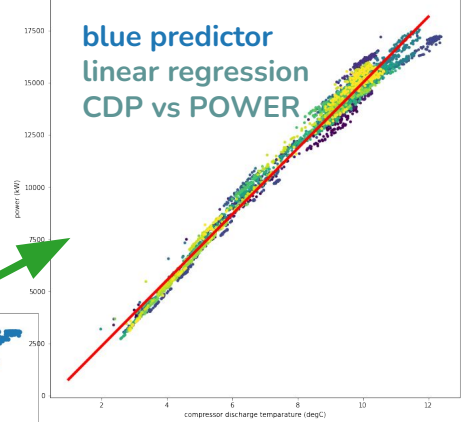




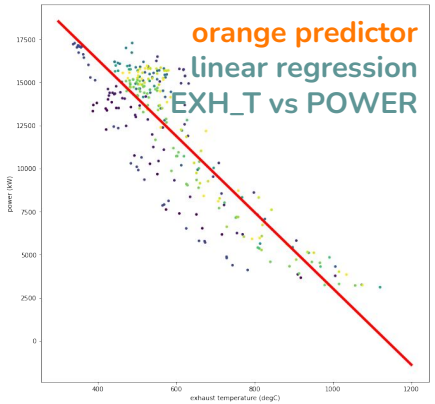
linear cut in POWER vs CDP



kNearNeigh in
CMP_SPEED
vs CDP



blue predictor
linear regression
CDP vs POWER



orange predictor
linear regression
EXH_T vs POWER

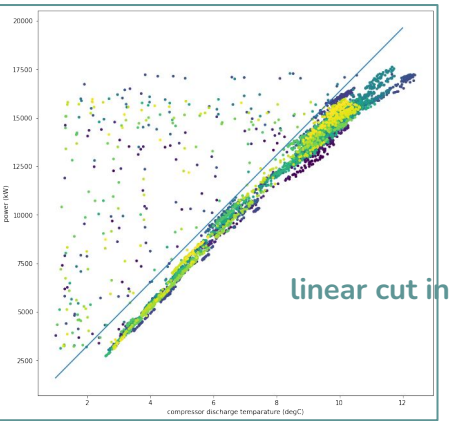
Implementation

For validation purposes, we focus in

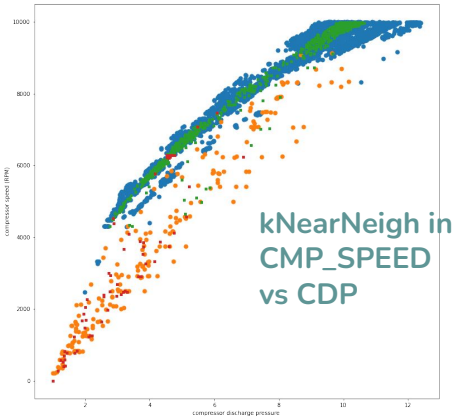
Hyperparameters:

linear cut
- ❑ slope parameter
- ❑ offset parameter

kNearNeigh
- ❑ k value



linear cut in POWER vs CDP



kNearNeigh in CMP_SPEED vs CDP

Implementation

For validation purposes, we focus in

Hyperparameters:

linear cut
- slope parameter
- offset parameter

kNearNeigh
- k value

EXH_T vs POWER linear regression
(Orange predictor)

CDP vs POWER linear regression
(Blue predictor)

```python
class MyModel():
    # default constructor
    def __init__(self, linear_cut_slope = 1640, linear_cut_intercept = 0, knum = 3):
        self.linear_cut_slope = linear_cut_slope
        self.linear_cut_intercept = linear_cut_intercept
        self.knum = knum

    # a method for printing data members
    def train(self, df_tr):
        self.df_tr = df_tr #dataframe for training

        # Linear Regression EXH_T vs POWER
        X_tr = np.array(df_tr['exhaust_temperature']).reshape(-1,1)
        Y_tr = np.array(df_tr['power']).reshape(-1,1)

        self.orange_model = LinearRegression().fit(X_tr, Y_tr)

        # KNeighbors data election with CDP and CMP_SPEED
        self.bluemask = df_tr['power'] < (self.linear_cut_slope)*df_tr['compressor_discharge_pressure'] + (self.linea
        df_X_kmeans = df_tr[['compressor_discharge_pressure', 'compressor_speed']].to_numpy()
        df_Y_kmeans = self.bluemask #np.array([0 for i in df_X_kmeans])

        self.chooseblue = KNeighborsClassifier(self.knum)
        self.chooseblue.fit(df_X_kmeans, df_Y_kmeans)

        # Linear Regression CDP vs POWER
        X_tr = np.array(df_tr['compressor_discharge_pressure']).reshape(-1,1)
        Y_tr = np.array(df_tr['power']).reshape(-1,1)

        self.blue_model = LinearRegression().fit(X_tr[self.bluemask], Y_tr[self.bluemask])

    def predict(self, df_pr): #dataframe for predicting

        # Predict with Linear Regression EXH_T vs POWER
        X_prt = np.array(df_pr['exhaust_temperature']).reshape(-1,1)
        Y_prt = self.orange_model.predict(X_prt)

        # Predict with Linear Regression CDP vs POWER
        X_prt1 = np.array(df_pr['compressor_discharge_pressure']).reshape(-1,1)
        Y_prt1 = self.blue_model.predict(X_prt1)

        # Blue or orange prediction with KNeighbors data
        self.bluelabel = self.chooseblue.predict(df_pr[['compressor_discharge_pressure', 'compressor_speed']].to_nump

        #X_prt = np.array(df_pr['compressor_discharge_pressure']).reshape(-1,1)
        #Y_prt = self.blue_model.predict(X_prt)
        for i in np.arange(len(Y_prt)):
            if (self.bluelabel[i] == True):
                Y_prt[i] = Y_prt1[i]

        for i in np.arange(len(Y_prt)):
            if ((df_pr['exhaust_temperature']==0)[i]==True):
                Y_prt[i] = 0

        return Y_prt
```

Implementation

For validation purposes, we focus in

Hyperparameters:

linear cut
- slope parameter
- offset parameter

kNearNeigh
- k value

EXH_T vs POWER linear regression (Orange predictor)

CDP vs POWER linear regression (Blue predictor)

```python
class MyModel():
    # default constructor
    def __init__(self, linear_cut_slope = 1640, linear_cut_intercept = 0, knum = 3):
        self.linear_cut_slope = linear_cut_slope
        self.linear_cut_intercept = linear_cut_intercept
        self.knum = knum

    # a method for printing data members
    def train(self, df_tr):
        self.df_tr = df_tr #dataframe for training

        # Linear Regression EXH_T vs POWER
        X_tr = np.array(df_tr['exhaust_temperature']).reshape(-1,1)
        Y_tr = np.array(df_tr['power']).reshape(-1,1)

        self.orange_model = LinearRegression().fit(X_tr, Y_tr)

        # KNeighbors data election with CDP and CMP_SPEED
        self.bluemask = df_tr['power'] < (self.linear_cut_slope)*df_tr['compressor_discharge_pressure'] + (self.linea
        df_X_kmeans = df_tr[['compressor_discharge_pressure', 'compressor_speed']].to_numpy()
        df_Y_kmeans = self.bluemask #np.array([0 for i in df_X_kmeans])

        self.chooseblue = KNeighborsClassifier(self.knum)
        self.chooseblue.fit(df_X_kmeans, df_Y_kmeans)

        # Linear Regression CDP vs POWER
        X_tr = np.array(df_tr['compressor_discharge_pressure']).reshape(-1,1)
        Y_tr = np.array(df_tr['power']).reshape(-1,1)

        self.blue_model = LinearRegression().fit(X_tr[self.bluemask], Y_tr[self.bluemask])

    def predict(self, df_pr): #dataframe for predicting

        # Predict with Linear Regression EXH_T vs POWER
        X_prt = np.array(df_pr['exhaust_temperature']).reshape(-1,1)
        Y_prt = self.orange_model.predict(X_prt)

        # Predict with Linear Regression CDP vs POWER
        X_prt1 = np.array(df_pr['compressor_discharge_pressure']).reshape(-1,1)
        Y_prt1 = self.blue_model.predict(X_prt1)

        # Blue or orange prediction with KNeighbors data
        self.bluelabel = self.chooseblue.predict(df_pr[['compressor_discharge_pressure', 'compressor_speed']].to_nump

        #X_prt = np.array(df_pr['compressor_discharge_pressure']).reshape(-1,1)
        #Y_prt = self.blue_model.predict(X_prt)
        for i in np.arange(len(Y_prt)):
            if (self.bluelabel[i] == True):
                Y_prt[i] = Y_prt1[i]

        for i in np.arange(len(Y_prt)):
            if ((df_pr['exhaust_temperature']==0)[i]==True):
                Y_prt[i] = 0

        return Y_prt
```

Implementation

For validation purposes, we focus in

Hyperparameters:

linear cut

- slope parameter
- offset parameter

kNearNeigh

- k value

determination of slope parameter and k value

```
num_folds = 4
k_choices = [1, 2, 3, 5, 8, 10]
intercept_choices = [-250, -125, 0, 125, 250, 375]


df_train_folds = np.array_split(df_train, 4)
accuracies = {}
```

```
for intercept_par in intercept_choices:
    for k in k_choices:
        accuracies[(k,intercept_par)] = []

        for i_folds in np.arange(num_folds):

            classifier = MyModel(linear_cut_intercept = intercept_par, knum = k)

            df_train_foldset = pd.concat(df_train_folds[:i_folds] + df_train_folds[i_folds+1:])
            df_test_foldset = df_train_folds[i_folds]

            classifier.train(df_train_foldset)

            y_val_pred = classifier.predict(df_test_foldset)

            MSE = mean_squared_error(df_test_foldset['power'], y_val_pred)
            RMSE = math.sqrt(MSE)

            #print(RMSE)
            accuracies[(k,intercept_par)].append(RMSE)
```

Implementation

For validation purposes, we focus in

Hyperparameters:

linear cut
- slope parameter
- offset parameter

kNearNeigh
- k value

determination of slope parameter and k value

```
for intercept_par in intercept_choices:
    for k in k_choices:
        print(accuracies[(k,intercept_par)], k,intercept_par)
```

```
[1099.1346176699706, 1186.1864374990985, 458.33667232959084, 801.0373755391048] 1 -250
[1176.1748988136167, 1213.7415163514963, 506.85154457416917, 753.791525556725] 2 -250
[1104.2329908659556, 1180.9054821146892, 560.7621590359508, 792.3219989650157] 3 -250
[1114.9724318738902, 1093.1364212209833, 633.0782248148932, 853.0426108837495] 5 -250
[1100.1131734566654, 1101.7766370853446, 621.17422362265, 852.2516153111765] 8 -250
[1092.2061351219018, 1119.9829962622762, 616.6459600142534, 849.1563659031169] 10 -250
[1081.6759188698131, 1185.5624089401147, 438.5999500526191, 800.0040307516825] 1 -125
[1154.3469506957956, 1199.4709511826977, 464.32929221348894, 750.1794890841571] 2 -125
[1104.5145435753025, 1180.84677577633, 559.3149696514487, 790.6950726109642] 3 -125
[1115.1828735946954, 1196.1396277874614, 631.5002747963688, 851.2217841515505] 5 -125
[1100.338258440857, 1101.7585067040288, 619.7109162712446, 850.4487903815495] 8 -125
[1092.2644702488415, 1119.8259609198958, 615.1874323053725, 847.370358769189] 10 -125
[1081.7867637526078, 1186.685521908773, 454.67356996199936, 797.7719730110771] 1 0
[1146.515791666771, 1196.0523600391716, 468.2470341748157, 746.207629272733] 2 0
[1082.9254787367806, 1182.8656932993867, 558.7891073304942, 789.1693884474179] 3 0
[1080.7074527189654, 1195.8246241060758, 630.9101300592719, 850.3450399238479] 5 0
[1074.653245249847, 1101.524701897947, 619.0613107872853, 849.5852925883783] 8 0
[1092.2985382495679, 1119.52273699411, 614.8375943342785, 846.5238246041997] 10 0
[1081.977286894574, 1179.3715769984947, 451.5347892955023, 797.345678280727] 1 125
[1145.5121221352485, 1188.816498928783, 465.1986333215167, 745.7841100469457] 2 125
[1083.0766227772924, 1182.5289238978926, 558.3975889079236, 788.7103312182285] 3 125
[1080.8191851825432, 1195.4751451695336, 630.4739720090686, 849.8754772881183] 5 125
[1074.7749338202311, 1101.2209952493592, 618.6474847040955, 849.1203610756321] 8 125
[1092.3509084046932, 1119.192090871778, 614.4254229527345, 846.0628320242532] 10 125
[1082.0567812687184, 1190.6552628236955, 451.41525188481995, 797.345678280727] 1 250
[1145.594077099688, 1188.5044566679942, 465.07768580756243, 745.7841100469457] 2 250
[1083.1399561255516, 1182.1923411450116, 558.2616498450374, 788.7103312182285] 3 250
[1080.8688021659837, 1195.1355078696163, 630.3270210066001, 849.8754772881183] 5 250
[1074.8278052028033, 1100.906158631533, 618.5012488295854, 849.1203610756321] 8 250
[1092.3752639269828, 1118.872441372991, 614.2814750423031, 846.0628320242532] 10 250
[1082.187787012699, 1189.9025055311508, 449.7990438018377, 796.820296551468] 1 375
[1145.7484285694861, 1187.8408412467832, 463.5041476382327, 744.1664981457147] 2 375
[1083.246560962938, 1181.4159354750084, 557.7799779418941, 788.0976953718168] 3 375
[1080.9396296817363, 1194.3357614907202, 629.7872965914244, 847.991461198329] 5 375
[1074.9089382959046, 1100.1999504765781, 617.988468225859, 848.5440958314767] 8 375
[1092.4133346102672, 1118.1178637687, 613.7719621186765, 845.4986052978034] 10 375
```

Implementation

For validation purposes, we focus in

Hyperparameters:

linear cut
- ❑ slope parameter
- ❑ offset parameter

kNearNeigh
- ❑ k value

k value = 3
offset parameter = 125

determination of slope parameter and k value

```
for intercept_par in intercept_choices:
    for k in k_choices:
        print(accuracies[(k,intercept_par)], k,intercept_par)
```

```
[1099.1346176699706, 1186.1864374990985, 458.33667232959084, 801.0373755391048] 1 -250
[1176.1748988136167, 1213.7415163514963, 506.85154457416917, 753.791525556725] 2 -250
[1104.2329908659556, 1180.9054821146892, 560.7621590359508, 792.3219989650157] 3 -250
[1114.9724318738902, 1093.1364212209833, 633.0782248148932, 853.0426108837495] 5 -250
[1100.1131734566654, 1101.7766370853446, 621.17422362265, 852.2516153111765] 8 -250
[1092.2061351219018, 1119.9829962622762, 616.6459600142534, 849.1563659031169] 10 -250
[1081.6759188698131, 1185.5624089401147, 438.5999500526191, 800.0040307516825] 1 -125
[1154.3469506957956, 1199.4709511826977, 464.32929221348894, 750.1794890841571] 2 -125
[1104.5145435753025, 1180.84677577633, 559.3149696514487, 790.6950726109642] 3 -125
[1115.1828735946954, 1196.1396277874614, 631.5002747963688, 851.2217841515505] 5 -125
[1100.338258440857, 1101.7585067040288, 619.7109162712446, 850.4487903815495] 8 -125
[1092.2644702488415, 1119.8259609198958, 615.1874323053725, 847.370358769189] 10 -125
[1081.7867637526078, 1186.685521908773, 454.67356996199936, 797.7719730110771] 1 0
[1146.515791666771, 1196.0523600391716, 468.2470341748157, 746.207629272733] 2 0
[1082.9254787367806, 1182.8656932993867, 558.7891073304942, 789.1693884474179] 3 0
[1080.7074527189654, 1195.8246241060758, 630.9101300592719, 850.3450399238479] 5 0
[1074.653245249847, 1101.524701897947, 619.0613107872853, 849.5852925883783] 8 0
[1092.2985382495679, 1119.52273699411, 614.8375943342785, 846.5238246041997] 10 0
[1081.977286894574, 1179.3715769984947, 451.5347892955023, 797.345678280727] 1 125
[1145.5121221352485, 1188.816498928783, 465.1986333215167, 745.7841100469457] 2 125
[1083.0766227772924, 1182.5289238978926, 558.3975889079236, 788.7103312182285] 3 125
[1080.8191851825432, 1195.4751451695336, 630.4739720090686, 849.8754772881183] 5 125
[1074.7749338202311, 1101.2209952493592, 618.6474847040955, 849.1203610756321] 8 125
[1092.3509084046932, 1119.192090871778, 614.4254229527345, 846.0628320242532] 10 125
[1082.0567812687184, 1190.6552628236955, 451.41525188481995, 797.345678280727] 1 250
[1145.594077099688, 1188.5044566679942, 465.07768580756243, 745.7841100469457] 2 250
[1083.1399561255516, 1182.1923411450116, 558.2616498450374, 788.7103312182285] 3 250
[1080.8688021659837, 1195.1355078696163, 630.3270210066001, 849.8754772881183] 5 250
[1074.8278052028033, 1100.906158631533, 618.5012488295854, 849.1203610756321] 8 250
[1092.3752639269828, 1118.872441372991, 614.2814750423031, 846.0628320242532] 10 250
[1082.187787012699, 1189.9025055311508, 449.7990438018377, 796.820296551468] 1 375
[1145.7484285694861, 1187.8408412467832, 463.5041476382327, 744.1664981457147] 2 375
[1083.246560962938, 1181.4159354750084, 557.7799779418941, 788.0976953718168] 3 375
[1080.9396296817363, 1194.3357614907202, 629.7872965914244, 847.991461198329] 5 375
[1074.9089382959046, 1100.1999504765781, 617.988468225859, 848.5440958314767] 8 375
[1092.4133346102672, 1118.1178637687, 613.7719621186765, 845.4986052978034] 10 375
```

Application

Post-processing

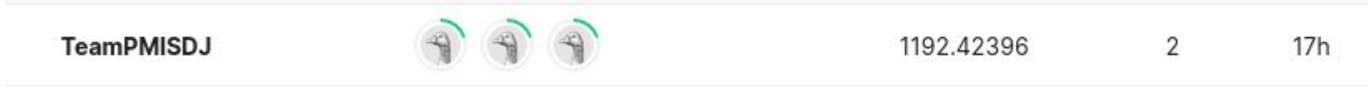
```
# Cambiar el tipo de la variable 'date' a tipo fecha
df_pred_output['date'] = pd.to_datetime(df_pred_output['date'])
```

```
# Modificar la columna de 'date' para que tenga el formato correcto de output
df_pred_output['date'] = df_pred_output['date'].dt.strftime('%-m/%-d/%Y')
```

```
#Guardar predicciones como submission.csv en esta carpeta
df_pred_output.to_csv('submission.csv', index = False)
```

Change date to correct type

Application



What could be improved?

- ❏ More submissions
- ❏ NaN values not considered at the beginning
- ❏ test_data_123.csv has a lot of NaN variables in the **EXH_T** variable, which does not happen in the trainig data, and was fundamental for the method implementation.a
- ❏ More investigation relating the turbines

Conclusions

- A ‘power’ estimation method for the turbines have been developed based on the Machine Learning classical algorithm (k-NearNeigh, LinearReg)
- The method returns values that vary wildly with the dataset (and not so much with the hyperparameters)
- This approach relies on separating into two classes (the **blue** and **orange** ones), in training and using **two different estimators** (both based on LinerReag) at predicting stage.
- The method suffered of the absence of data in the test_data_123.csv file, particularly in the **EXH_T** variable.

Thanks!