i) Using Seaborn's heatmap function, create a heatmap that clearly shows the correlations (R) for all variables using the dataframe_raw dataset.

The syntax of Seaborn takes the following arguments:

sns.heatmap(correlated_dataframe, annot=True)

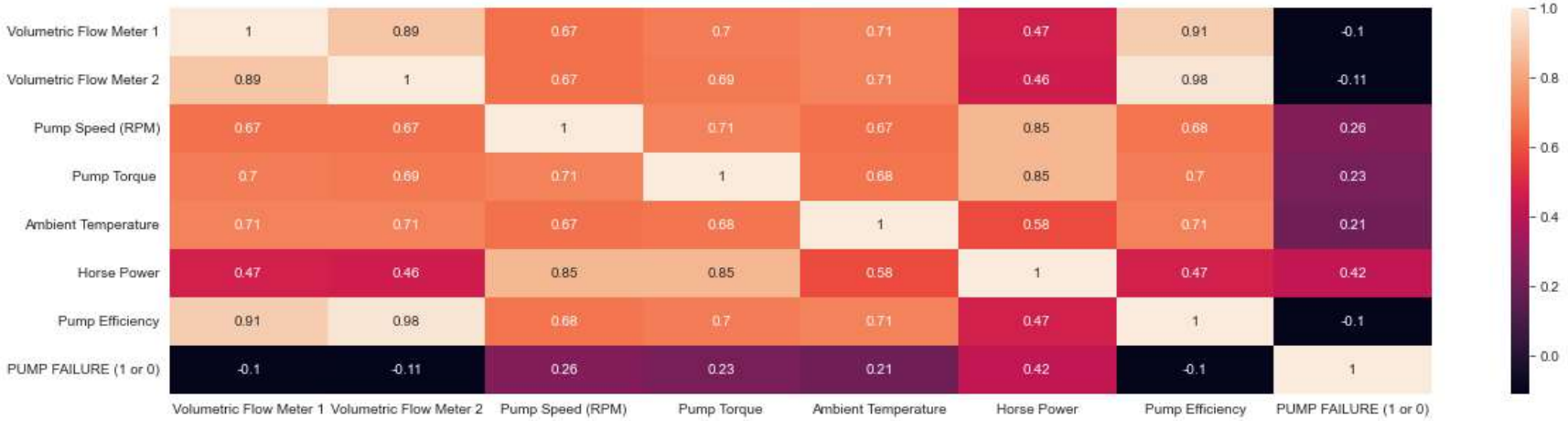
You'll need to ensure you use the following .corr() function and apply this to your dataframe first, before passing this to the sns.heatmap() function.

Please put your code here

We've included an example of what the output *may* look like below

In [70]:
```
heatmap1=dataframe_raw.corr(method='pearson').round(decimals=2)
sns.heatmap(heatmap1,annot=True)
```

Out[70]:
```
<AxesSubplot:>
```



Which variables seem to correlate with Pump Failure?

In [71]:
```
#The best correlation number Pump failure has with is horse power.
```

Step 11: Create a Barplot of Correlated Features

i) Using the correlated DataFrame you created earlier, create a barplot that shows the correlated features against PUMP FAILURE (1 or 0), in descending order.

You'll have to think carefully regarding this question.

You'd know that you can get the correlation values from the following command:

```
dataframe.corr()
```

Upon printing this out, you should get a DataFrame that looks like the below:

In [72]:
```
heatmap1
```

Out[72]:

	Volumetric Flow Meter 1	Volumetric Flow Meter 2	Pump Speed (RPM)	Pump Torque	Ambient Temperature	Horse Power	Pump Efficiency	PUMP FAILURE (1 or 0)
Volumetric Flow Meter 1	1.00	0.89	0.67	0.70	0.71	0.47	0.91	-0.10
Volumetric Flow Meter 2	0.89	1.00	0.67	0.69	0.71	0.46	0.98	-0.11
Pump Speed (RPM)	0.67	0.67	1.00	0.71	0.67	0.85	0.68	0.26
Pump Torque	0.70	0.69	0.71	1.00	0.68	0.85	0.70	0.23
Ambient Temperature	0.71	0.71	0.67	0.68	1.00	0.58	0.71	0.21
Horse Power	0.47	0.46	0.85	0.85	0.58	1.00	0.47	0.42
Pump Efficiency	0.91	0.98	0.68	0.70	0.71	0.47	1.00	-0.10
PUMP FAILURE (1 or 0)	-0.10	-0.11	0.26	0.23	0.21	0.42	-0.10	1.00

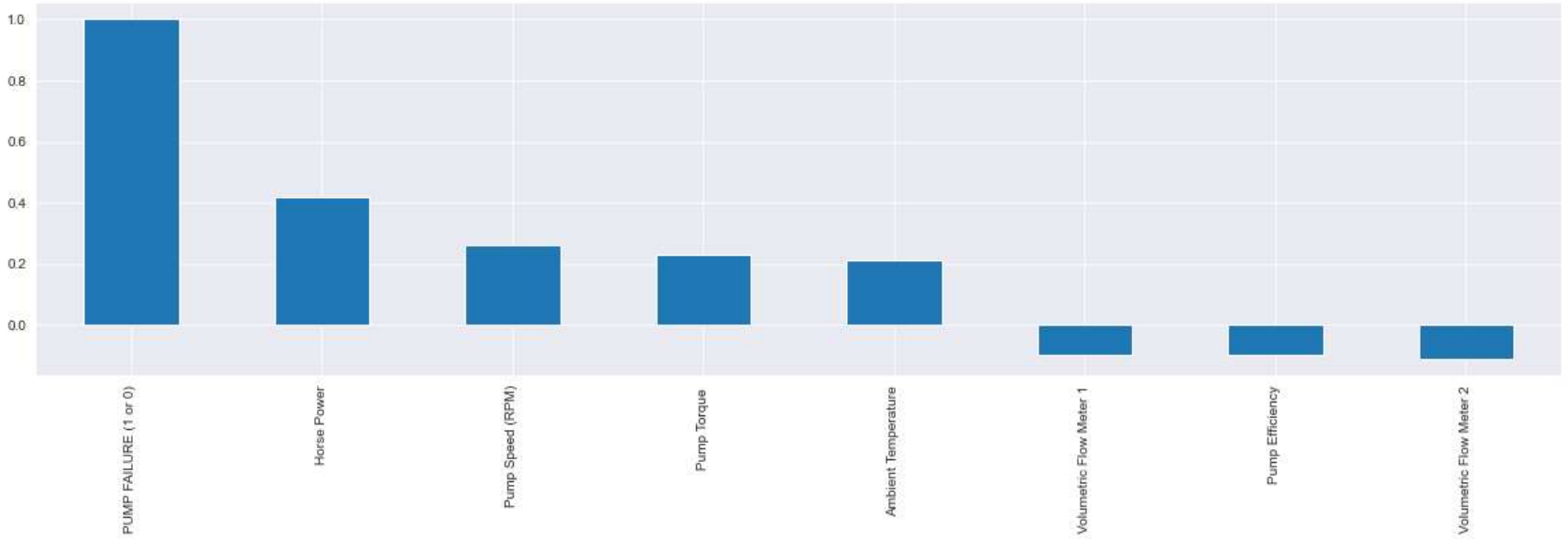
You might think to yourself this looks suspiciously like a DataFrame with columns that you can manipulate.

You can.

With this hint in mind, you should be able to select the relevant column of interest and combine this with the .plot(kind='__') function to answer the question. Don't forget to think about .sort_values() to help sort your data!

Please put your code here

In [73]:
```
heatmap1['PUMP FAILURE (1 or 0)'].sort_values(ascending=False).plot(kind='bar')
plt.show()
```



Step 12: Create a Rolling Standard Deviation Heatmap

Previously, you created a correlation matrix using 'raw' variables. We saw *some* correlations with the raw data but they weren't necessarily as strong as we would have liked. This time, we'll recreate a Heatmap using the dataframe_stdev dataset you had imported in Step Two.

i) Using Seaborn's heatmap function, create a heatmap that clearly shows the correlations (including R) for all variables using the dataframe_stdev dataset.

Please put your code here

In [58]:
```
sns.heatmap(dataframe_stdev.corr(), annot=True)
```

Out[58]:
```
<AxesSubplot:>
```

Do any variables stand out? If yes, list these out below your heatmap.

In [60]:
```
print("Horse power is the highest at .69, however all the variables have strong correlation.")
```

```
Horse power is the highest at .69, however all the variables have strong correlation
```

Creating a Multivariate Regression Model

When you worked on this case study in Excel, you went through the tricky process of using the rolling standard deviation variables to generate a regression equation. Happily, this process is much simpler in Python.

For this step, you'll be using the statsmodel.api library you imported earlier and calling the Ordinary Least Squares Regression to create a multivariate regression model (which is a linear regression model with more than one independent variable).

Step 13: Use OLS Regression

i) Using the OLS Regression Model in the statsmodel.api library, create a regression equation that models the Pump Failure (Y-Variable) against all your independent variables in the dataframe_raw dataset.

In order to fit a linear regression model with statsmodels.api there are a few steps that need to be taken. We have demonstrated this below:

Don't forget to reimport the DataFrames you've previously imported in Step 2 before starting these steps

1. Establish two DataFrames named, independent_variables and dependent_variables. The independent variables are known as explanatory variables - they help EXPLAIN what you are trying to model. Dependent Variable on the other hand is the variable of interest that you want to MODEL. In this case, the Dependent Variable is Pump Failure (1 or 0).

2. Add a constant to your Independent Dataframe via the following syntax:
independent_variables = sm.add_constant(independent_variables). This will simply add a constant stream of 1's in a column to your dataframe. This constant is used to account for bias in the model.

3. Store and Fit your model with the below syntax:
regression_model = sm.OLS(dependent_variable,independent_variable).fit()

4. Print the regression_model.summary() to view the Regression Statistics

ii) Repeat i) but this time use the dataframe_stdev you imported previously.

You will repeat the same steps as you have done in i) only you will be changing the dataset from dataset_raw to dataset_stdev.

Please put your code here

In [50]:
```
#Dataframe_Raw
import numpy as np
import statsmodels.api as sm
X=dataframe_raw[['Volumetric Flow Meter 1',
       'Volumetric Flow Meter 2', 'Pump Speed (RPM)', 'Pump Torque ',
       'Ambient Temperature', 'Horse Power', 'Pump Efficiency']
]
X=sm.add_constant(X)
```

```
#X=X.astype(float)
Y=dataframe_raw['PUMP FAILURE (1 or 0)']
#Y=Y.astype(float)
regression_model=sm.OLS(Y, X).fit()
regression_model.summary()
```

Out[50]:

OLS Regression Results

Dep. Variable:	PUMP FAILURE (1 or 0)	R-squared:	0.362
Model:	OLS	Adj. R-squared:	0.360
Method:	Least Squares	F-statistic:	197.9
Date:	Sun, 17 Jul 2022	Prob (F-statistic):	5.58e-233
Time:	22:22:54	Log-Likelihood:	1823.0
No. Observations:	2453	AIC:	-3630.
Df Residuals:	2445	BIC:	-3583.
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.1138	0.028	4.058	0.000	0.059	0.169
Volumetric Flow Meter 1	-0.0145	0.002	-7.974	0.000	-0.018	-0.011
Volumetric Flow Meter 2	-0.0184	0.003	-5.497	0.000	-0.025	-0.012
Pump Speed (RPM)	0.0021	0.001	2.589	0.010	0.001	0.004
Pump Torque	0.0007	0.000	1.801	0.072	-6.24e-05	0.001
Ambient Temperature	0.0099	0.001	14.159	0.000	0.009	0.011
Horse Power	0.0827	0.019	4.373	0.000	0.046	0.120
Pump Efficiency	0.0020	0.002	1.028	0.304	-0.002	0.006

Omnibus:	2071.669	Durbin-Watson:	0.458
Prob(Omnibus):	0.000	Jarque-Bera (JB):	58977.621
Skew:	3.929	Prob(JB):	0.00
Kurtosis:	25.700	Cond. No.	3.03e+03

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.03e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In [51]:
```
#Dataframe_Stdev
X=dataframe_stdev[['Volumetric Flow Meter 1',
       'Volumetric Flow Meter 2', 'Pump Speed (RPM)', 'Pump Torque ',
       'Ambient Temperature', 'Horse Power', 'Pump Efficiency']
]
X=sm.add_constant(X)
#X=X.astype(float)
Y=dataframe_stdev['PUMP FAILURE (1 or 0)']
#Y=Y.astype(float)
regression_model=sm.OLS(Y, X).fit()
regression_model.summary()
```

Out[51]:

OLS Regression Results

Dep. Variable:	PUMP FAILURE (1 or 0)	R-squared:	0.778
Model:	OLS	Adj. R-squared:	0.778
Method:	Least Squares	F-statistic:	1225.
Date:	Sun, 17 Jul 2022	Prob (F-statistic):	0.00
Time:	22:23:07	Log-Likelihood:	3117.4
No. Observations:	2452	AIC:	-6219.
Df Residuals:	2444	BIC:	-6172.
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0354	0.004	-10.045	0.000	-0.042	-0.029
Volumetric Flow Meter 1	0.0513	0.010	5.109	0.000	0.032	0.071
Volumetric Flow Meter 2	-0.6085	0.020	-30.450	0.000	-0.648	-0.569
Pump Speed (RPM)	-0.0178	0.002	-9.057	0.000	-0.022	-0.014
Pump Torque	-0.0189	0.001	-22.511	0.000	-0.020	-0.017
Ambient Temperature	0.0178	0.003	5.773	0.000	0.012	0.024
Horse Power	0.7636	0.022	35.045	0.000	0.721	0.806
Pump Efficiency	0.3407	0.012	28.118	0.000	0.317	0.364

Omnibus:	1432.369	Durbin-Watson:	0.242
Prob(Omnibus):	0.000	Jarque-Bera (JB):	72092.923
Skew:	2.052	Prob(JB):	0.00
Kurtosis:	29.245	Cond. No.	426.

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Compare the Models you've created and observe the R Squared displayed for each model. Which model seems better and why? Put your thoughts down below. If you're not sure how to answer this, you should reference the resource here for a more detailed explanation.

In [55]:
```
print("The Stdev model seems like a better fit, because the std err is lower oeverall, so it is more likely to be corre
```

```
The Stdev model seems like a better fit, because the std err is lower oeverall, so it is more likely to be correct than
the raw data model. The R sqared value 0.3 for raw data is very week. Which meand this model is not significant to expl
ain the pump failure. But the R squared value 0.7 for rolling stdev data is very strong. This model can explain better.
```

Step 14: Analysis of Coefficients

Great job creating those regressive equations! You've reached the final two steps of this case study!

You've identified that *one* regressive model has a better fit than the previous model.

Using this model, we want to extract the co-efficients to better understand the variables that display the largest *absolute* rate of change with respect to Pump Failure. We'll use the co-efficients to better assess this from an *absolute* rate of change perspective. (i.e. Which variables spike the most in relation to Pump Failure?)

i) Using the .params method, extract the Coefficients from your regression_model and create a bar plot that identifies which coefficients react most strongly with respect to Pump Failure. Which three (3) variables seem to showcase the strongest 'relationship' with respect to Pump Failure?

You can use params via the following syntax:

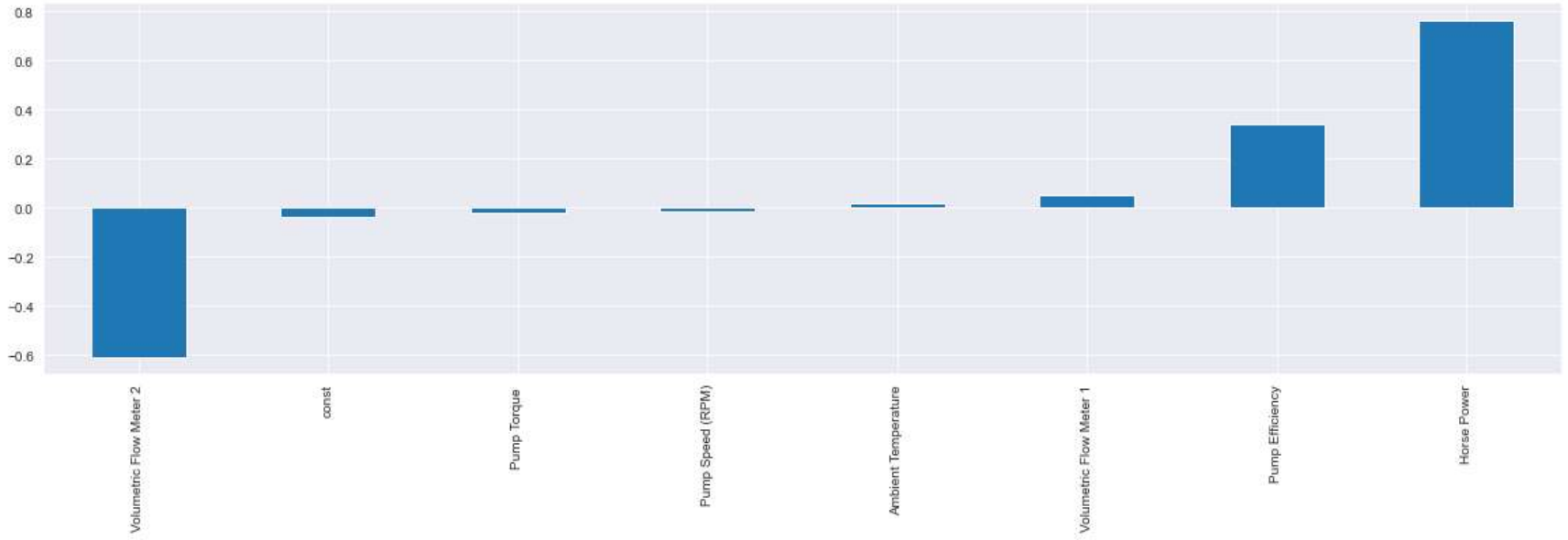
```
model.params where model represents the name of the regression model you've created.
```

We've included an example below that showcases what your chart should look like upon completion of this exercise.

Please put your code here

In [56]:
```
regression_model.params.sort_values(ascending=True).plot(kind='bar')
```

Out[56]:
```
<AxesSubplot:>
```



Please note that Step 15 is a Challenge Questions and will intentionally be more difficult

Step 15: Validate Predictions

Now it's time for us to validate our predictions.

Once you've created a regressive_model, you can call this using the following syntax:

```
regressive_model.predict(independent_variables).
```

Extra information regarding how this works can be found here

i) Create a new column in the dataframe_stdev, called, 'Prediction'.

ii) Use the regression equation you created in the previous step and apply the .predict() function to the independent variables in the dataframe_stdev dataset so you get a column full of your regressive predictions.

iii) Create a Dual-Axis Plot with the following axes items:

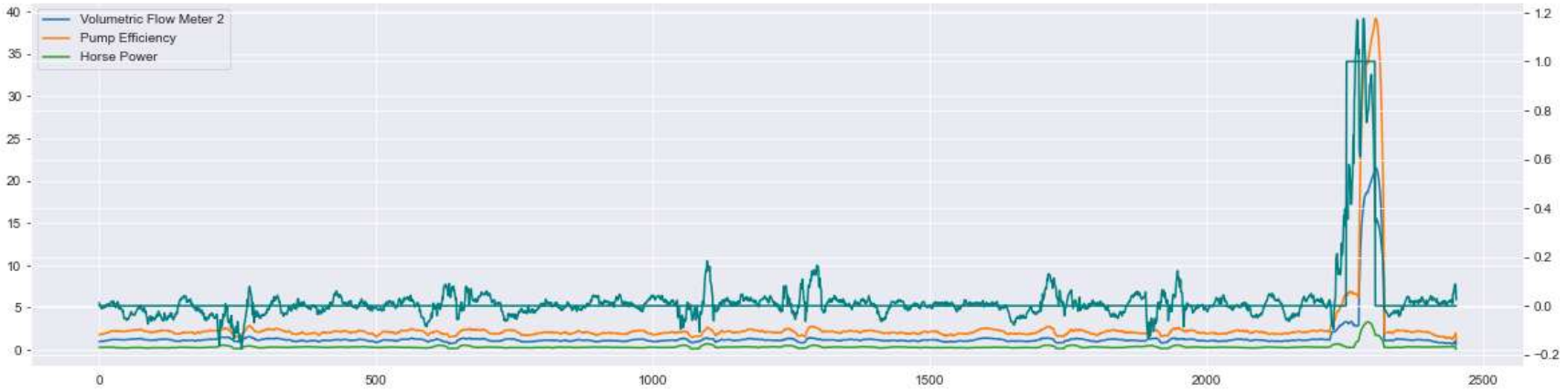
```
    Axes One would contain: Volumetric Flow Meter 2, Pump Efficiency and Horse Power
    Axes two would contain: Pump Failure (1 or 0) and Prediction
</b>
```

Note: Don't forget how to use .twinx() to help you out with the dual axis!

We've included an example below of how your chart might look once you're done!

Please put your code here

In [57]:
```
dataframe_stdev['Prediction']=regression_model.predict(X)
dataframe_stdev
first_axis = dataframe_stdev[['Volumetric Flow Meter 2', 'Pump Efficiency', 'Horse Power']].plot()#Looping through ever
second_axis = first_axis.twinx() #The Twinx function is used to ensure we share the X-Axis for both plots
second_axis.plot(dataframe_stdev[['PUMP FAILURE (1 or 0)', 'Prediction']], color='teal')
    #plt.title()
plt.show()
```



You've made it to the end of this challenging case study — well done!

You've learnt to perform a number of new opeartions in Python.

You're now able to:

- **Create Line Plots and Box Plots**
- **Understand Descriptive Statistics from the .describe() functions**
- **Detect and Remove Outliers from your Dataset**
- **Subset and Filter your DataFrames**
- **Use For Loops to smartly loop through data**
- **Use Dual Axes to Plot Multiple Variables on different axes**
- **Interpret Correlation Coefficients and Heatmaps**
- **Create and assess the goodness of fit for your Linear Regression Models**

Well done! This is just the start of your foundation in Python as you grow more proficient in practicing against datasets both big and small.

Keep up the fantastic work and as always - if you have any questions, don't hesitate to reach out to the community, your mentor, or #slack channel for extra support!

Keep up the amazing work!