Using Inline Test Data to Predict Final Product Quality Pass / Fail

Capstone Project of Advanced Data Science

Outline

- User Case Explanation
- Dataset Explanation
- Data Quality Assessment
- Data Exploration and Visualization
- Feature Engineering
- Model Selection and Comparison
- Conclusion

User Case Explanation

- Background: In a product manufacturing scenario, there are many process steps in a sequence to manufacture a product. As a product goes through all these process steps, multiple tests are taken to evaluate quality of these process steps. Such tests are called inline tests. After all process steps are finished, the product goes through a final function test to ensure it meets all the functional requirements. If we could build a model of using inline test results to predict final test's pass / fail, such model will provide tremendous benefits of detecting early fail issue (so the product can stop further processing and save cost) and reacting sooner for corrective actions.
- User Case: We would like to collect dataset of all inline tests' result and final functional test's result to creat such a prediction model.

Dataset Explanation

- The dataset is from a private company's manufacturing data source. To protect IP, all column names are renamed to generic indicators such as test 1,2,3, etc. and final result. Dataset is exported to csv file for ETL.
- The csv file is then uploaded to IBM cloud object storage and retrieved in jupyter notebook as pandas dataframe.

Data Quality Assessment

- 3 steps are taken to assess the data quality
  - Initial inspection: 1 column of sample ID; 252 columns of inline test result (features); 1 column of final test pass/fail (represented by value 1/0)
  - Convert all columns to float type for model building.
  - Inspect for NaN values. There are 100 NaN values. Delete those rows since we have >3000 rows (still enough sample size)
- After assessment,

Initial inspection

Examine the dataframe created from the csv file

We can see there are 254 columns. First it contains a sample ID column, then test results from 252 tests, and a final test result as pass (1) or fail (0). The goal is the build a model using the 252 tests to predict the final test result.

```
In [4]: df_data.head()
```

Out[4]:

	Sample_ID	Test_1	Test_2	Test_3	Test_4	Test_5	Test_6	Test_7	Test_8	Test_9	Test_10	Test_11	Test_12	Test_13	Test_14	Test_15	Tes
0	1	4.257	12.617	28.153	37.445	24.258	3.370	455.253	4361.820	3631.450	23.649	63.847	3.337	23.096	28.79	25.10	23.
1	2	4.291	13.027	28.213	38.438	24.337	3.416	444.421	4307.690	3771.835	24.230	63.828	3.405	23.676	30.90	24.66	22.
2	3	4.276	12.865	28.441	38.974	25.091	3.383	432.610	4247.070	3840.860	23.724	65.467	3.445	23.168	34.70	29.91	23.
3	4	4.292	12.788	27.937	39.390	25.011	3.441	449.512	3922.305	3868.475	23.838	64.755	3.492	23.289	33.31	28.94	23.
4	5	4.306	12.844	28.612	38.471	24.991	3.400	444.022	4268.240	3889.510	23.588	65.834	3.444	23.041	32.37	27.54	24.

We can also see we have 3702 rows of data

```
In [19]: df_data.shape
Out[19]: (3702, 254)
```

```
In [32]: df_data.isna().values.sum()
Out[32]: 100
```

Obseved 100 NaN values

```
In [5]: df_test.dropna(inplace=True)
        df_test.reset_index(drop=True,inplace=True)

In [6]: # double check there is no nan value
        df_test.isna().values.sum()
Out[6]: 0

In [40]: df_test.shape
Out[40]: (3652, 254)
```

Remove those NaN rows. Still have 3652 rows of data, enough for model building.

Data Exploration and Visualization

- 4 steps are taken for data exploration and visualization
  - General statistical metrics (using describe function): we can see 252 feature columns with values varying significantly between each other, indicating some standardization may be needed.
  - Correlation matrix generation and visualization with heatmap: we can see some features are correlating to each other, indicating PCA can be tried for feature dimention reduction.
  - Visualize correlation between features using scatter plot.
  - Visualize correlation between feature and final label (final test pass/fail) using box plot.

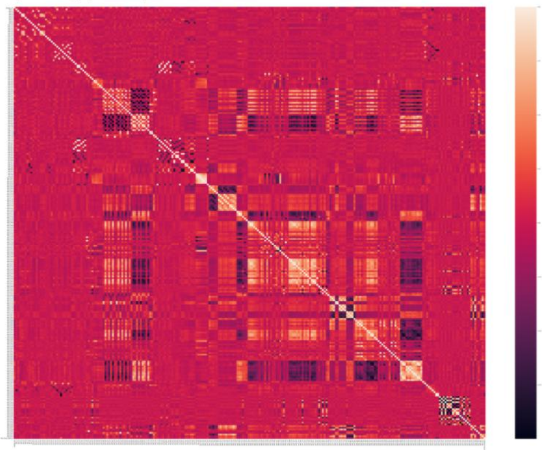
General statistical metrics

In [41]:
```
df_test.describe(include='all')
```
Out[41]:

	Sample_ID	Test_1	Test_2	Test_3	Test_4	Test_5	Test_6	Test_7	Test_8	Test_9	
count	3652.000000	3652.000000	3652.000000	3652.000000	3652.000000	3652.000000	3652.000000	3652.000000	3652.000000	3652.000000	3652
mean	1847.847755	4.455589	13.845736	27.743817	39.264878	25.661343	3.283326	453.511081	4194.747842	4011.318613	25.2
std	1068.668921	0.132352	0.519216	0.688015	1.104378	1.000117	0.073000	19.777875	268.899911	286.204673	0.87
min	1.000000	4.023000	12.398000	25.218000	35.193000	22.254000	3.076000	383.063000	3302.490000	3264.600000	23.0
25%	920.750000	4.365000	13.461000	27.270000	38.501000	24.960750	3.232000	440.237750	4011.511250	3833.658750	24.6
50%	1847.500000	4.449500	13.840000	27.766000	39.228000	25.613000	3.281000	452.663500	4201.575000	3989.425000	25.2
75%	2773.250000	4.548000	14.232000	28.201000	39.961000	26.311250	3.331000	466.253500	4362.762500	4155.785000	25.8
max	3702.000000	5.534000	15.427000	30.964000	43.438000	29.412000	3.565000	528.132000	5323.055000	6416.950000	28.5

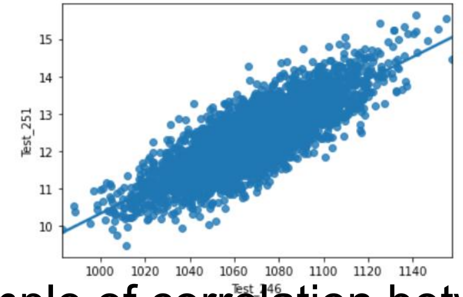
Correlation matrix

In [45]:
```
plt.figure(figsize=(80,60))
sns.heatmap(df_corr,
            xticklabels=df_corr.columns.values,
            yticklabels=df_corr.columns.values)
```
Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff199c4fc50>



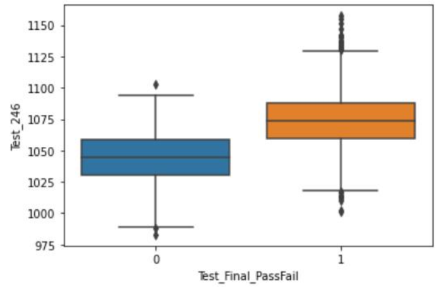
Example of correlation between 2 features

In [48]:
```
sns.regplot(data=df_data,x='Test_246',y='Test_251')
```
Out[48]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff18d977310>



Example of correlation between 1 feature and final label

In [49]:
```
sns.boxplot(data=df_data,x='Test_Final_PassFail',y='Test_246')
```
Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff199e36490>



Feature Engineering

- 2 steps are taken for feature engineering
  - Standardization using sklearn StandardScaler function: we observed significant (several orders of magnitude) value difference among feature columns, so we took standardization which is essential to improve accuracy of models of SVM and MLP.
  - PCA using sklearn PCA function: we observed correlation between feature columns from the correlation matrix, which indicates PCA could be tried to reduce feature dimension. While the model accuracy is worse with PCA components, training speed is faster. This provides reference for future model adjustment when we continue to have more features and need to make a trade-off between model accuracy and training speed.

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

Standardization

```
# Standardize the feature columns using the StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

PCA

```
In [88]: # Perform PCA on the X_scaled dataset
         pca = PCA()
         pca.fit(X)
Out[88]: PCA()
```

first 50 PCA components can already explain ~100% variance. So we can just take the first 50 components.

```
In [89]: (pca.explained_variance_ratio_[0:50]).sum()
Out[89]: 0.9999999999987866

In [95]: # Then set the PCA component number
         pca2 = PCA(n_components=50)
         pca2.fit(X)
Out[95]: PCA(n_components=50)

In [96]: X_pca = pca2.transform(X)
```

Model Selection and Comparison

- 3 models were built
  - Non-deep-learning Models
    - Gradient-Boosted Tree (GBT): This model is selected because literature suggests it will be very effective for binary classification problem
    - Support Vector Machine (SVM): This model is anothe commonly used model for binary classification
  - Deep-learning Model
    - Multiple Layer Perceptron (MLP): This model is a commonly used model for binary classification
- Model Performance Comparison
  - Metric: using F1-score as the metric to compare among models for our binary classification user case.
  - Compare among 3 models with and without feature engineering
  - Conclusion
    - GBT gives the best performance with or without feature engineering
    - Standardization clearly helps SVM / MLP's accuracy
    - PCA redues models' accuracy but improves training speed with redced feature size, providing a reference for future trade-off.

F1-score for model performance comparison

```
In [9]: df_perf.style.background_gradient(cmap='Greens')
Out[9]:
```

	Feature Engineering	Model 1: GBT	Model 2: SVM	Model 3: MLP
0	None	0.956000	0.906000	0.906000
1	Standardization	0.956000	0.947000	0.951000
2	PCA + Standardization	0.916000	0.916000	0.905000

Conclusion

- Comparing all the model cases (3 models, with standardization, with PCA), the GBT model with standardization and learning rate of 0.25 gives the best F1-score of 0.956. This would be the model we will deploy for predicating the final test pass / fail.