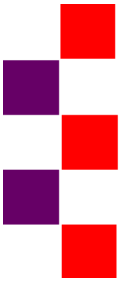


University of Essex

Design and Application of a Machine Learning System for a Practical Problem

CE802 Machine Learning by Dr Vito De Feo

Investigative Report for AENERGY

Submission by: Abdullah Mujeeb Khawaja - 2201728

Word Count: ~2K

AENERGY's task for predicting customer's ability to pay the bill

Classification Task

Problem Statement

AENERGY wants to find out what customers under their service will suffer due to the rising cost of electricity bills and not manage to pay them as a result. This is a classification problem, and we will help support the organization by harnessing the ability given to us by Machine Learning.

Data Understanding

We start diving into the data straight away that is given to us by AENERGY; We are given 21 features and a class distinguishing users from those who have paid up historically and those who haven't. Our data is about 1000 rows and continuous. We also learn that only one feature, F21, has missing values.

We can see from Figure 1 that F23, the one with missing values, is heavily negatively correlated with our target variable, and so the importance is right away established. It translates to mean that the impact of F23 on our Class is inverse in relationship.
Also, we learn that F3, F12, and F1 are positively correlated with our target variable. It would be interesting to revisit this when we will move forward and extract feature importance to see whether our models establish a similar feature importance or not.

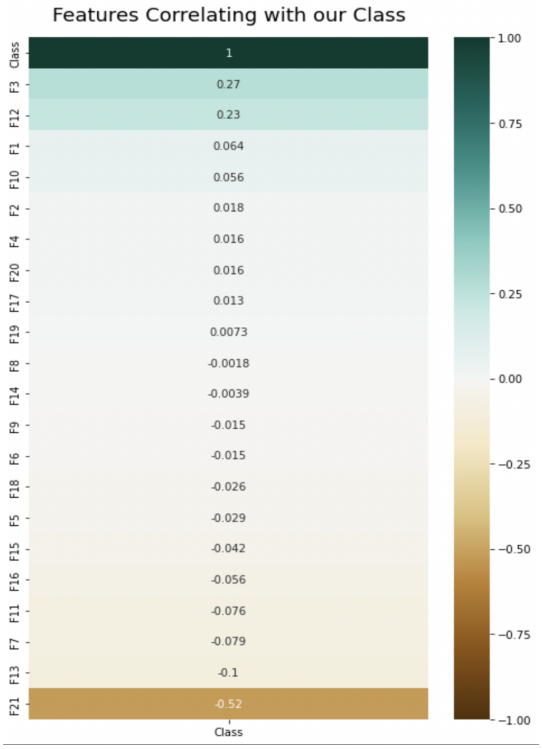
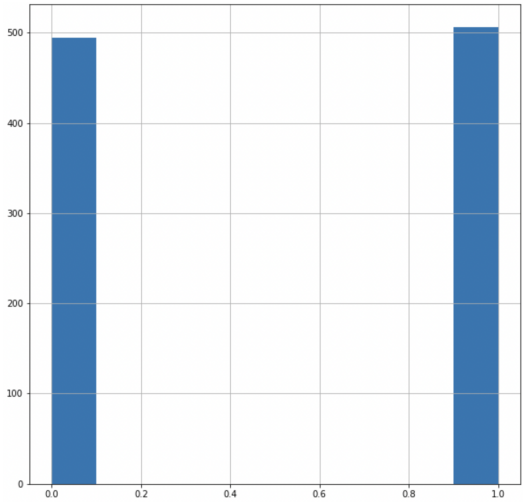


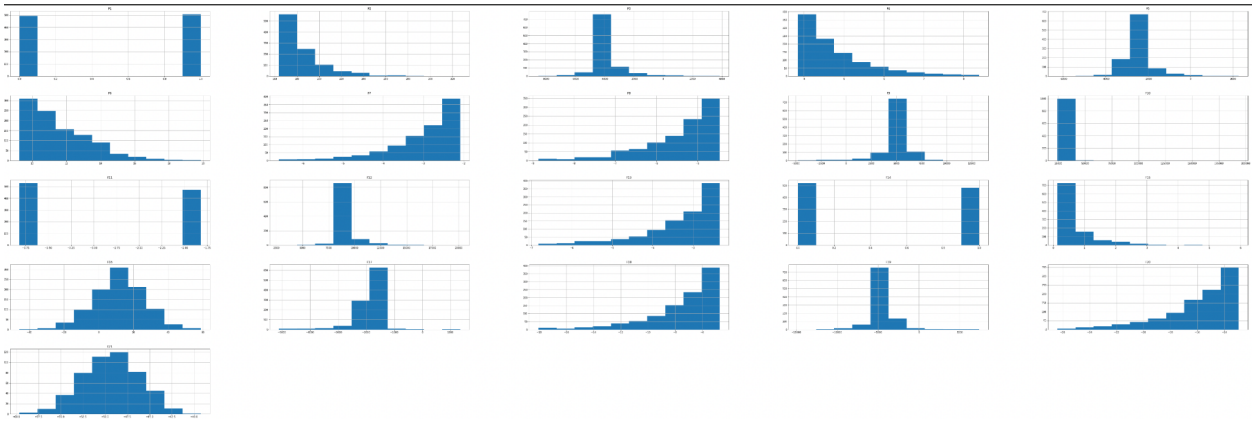
Figure 1: Class-Feature

With our second figure, we see the Class distribution and note that they are balanced, which is a great thing because it enables us to avoid using techniques like SMOTE/GANs/Data Augmentation to combat imbalance datasets.

Balanced dataset also enables us to look at accuracy as an evaluation metric and not make use of class_weights in sklearn for artificially balancing the effect of the dominant class.

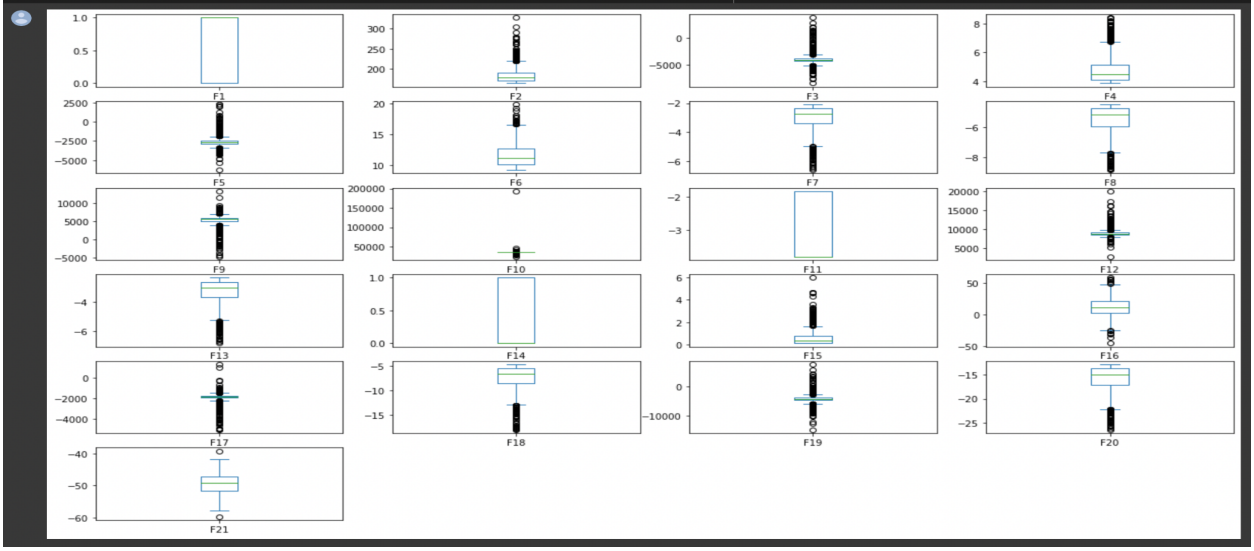


Next, we view histograms and box plots for our feature set to further study our data



We can see that our data is skewed in most places and normal in others. While F23 seemingly is normal, I would not risk imputing it with just the mean, rather I would extend my approach and try out different imputation methods and combine them with different normalization methods and combine that also with various selective outlier methods and then run the models across all variants to choose the best variant. At the end of the day, data is the new oil, and what it tells us, we accept after testing rigorously.

```
#Furthering the effort of outlier detection, lets use box plots.
AENERGY.plot(kind='box', subplots=True, figsize=(20, 40),layout=(20,4))
plt.show()
#While we can see outliers present, it is better to do this again after having normalized columns so ranges are better.
```



From above, we can gauge that outliers are most definitely present, especially evident in say the cae of F10. This was also obvious when the mean and max difference was marginal when dataframe was described.

Approach & Model Training

- Apply all algorithms on default with missing values dropped
- Apply data transformation and handle missing values
- Apply all algorithms on default and utilize GridSearchCV and pipeline for best models
- Ensemble them for final prediction.

Lets start with a Decision Tree classifier, nothing fancy, no finetuning, just straight away testing what it would do with our data

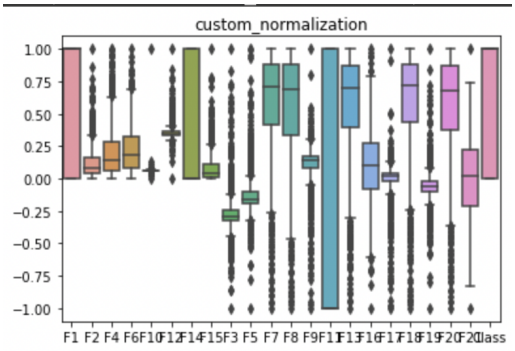
```
[ ] df=AENERGY.copy(deep=True)
    X_train, X_test, y_train, y_test = train_test_split(df.drop(columns="Class"), df["Class"], test_size=0.3)
    model = DecisionTreeClassifier(random_state=42)
    model.fit(X_train, y_train)
    train_score = model.score(X_train,y_train)
    print("Train set score:", round(train_score,3))
    test_score = model.score(X_test, y_test)
    print("Test set score:", round(test_score,3))

    Train set score: 1.0
    Test set score: 0.747
```

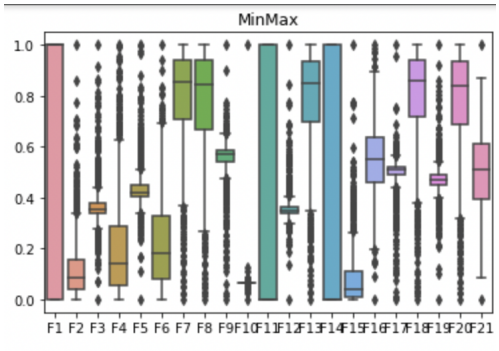
Right away with the Decision Tree on raw dataset, we see that we are already overfitting heavily.
Let's try one more and see if we are moving in the same direction.

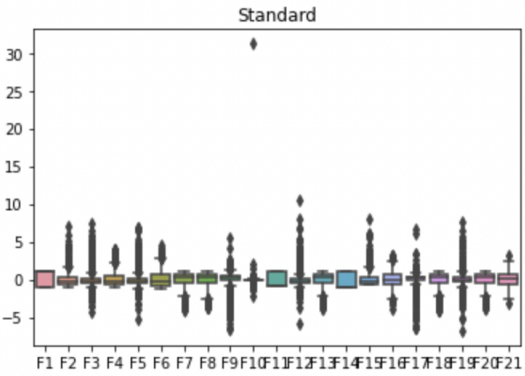
Now, we will try KNN, another classification algorithm that can utilize the count of attributes matching for figuring out the discrimination factor.

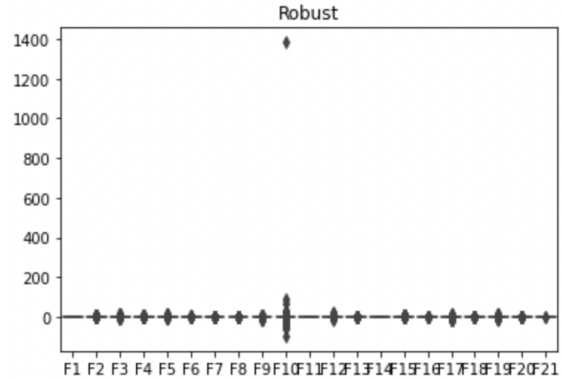
```
knn_model = KNeighborsClassifier()
knn_model.fit(X_train, y_train)
train_score = knn_model.score(X_train,y_train)
print("Train set score:", round(train_score,2))
test_score = knn_model.score(X_test, y_test)
print("Test set score:", round(test_score,3))

Train set score: 0.73
Test set score: 0.58
```

We learn the importance of feature normalization. We try out Custom Normalization but drop it early, and try our Standard Scaler, Robust, and Min-Max. Robust tackles outliers on its own.









With custom normalization, it was my aim to normalize between -1,1 for features that had negative values using MinMax, and 0,1 for features that were positive. And we see that Standard and Robust is already the better looking. However, for the sake of experimentation, we will maintain a dictionary of dataframes and run outlier methods on all 3 normalizations; MinMax, Standard, Robust.

At the same time, we will impute using 3 methods; **KNNImputer**, **SimpleImputer** with *Mean* and *Median*.

Given our data has a box plot with all features well outside of the box on average, it could indicate that there are a decent number of anomalies(we already know our classes are balanced hence this phenomena) present in the data. In this case:

- **Isolation Forest**: Isolation Forest is an unsupervised machine learning algorithm that is used to identify outliers in the data. It works by randomly selecting a feature and then randomly selecting a split value between the minimum and maximum values of the selected feature. The algorithm then splits the dataset into smaller and smaller subsets, and assigns an anomaly score to each record based on the number of splits required to isolate it.

- **Local Outlier Factor** (LOF): The Local Outlier Factor (LOF) is an unsupervised machine learning algorithm that is used to identify outliers in the data. The algorithm assigns an outlier score to each record, with higher scores indicating that the record is more likely to be an outlier.

- **Robust covariance Estimator**: it is used to detect the outliers in the data by assuming that the data follows a Gaussian distribution and identifying the observations that deviate from this distribution. It is less sensitive to the presence of outliers in the data than the traditional covariance estimator.

```
outliers_isolation = set(df_imputed[key].index.difference(df_no_outliers['Isolation Forest '+ key].index))
outliers_lof = set(df_imputed[key].index.difference(df_no_outliers['Local Outlier Factor '+ key].index))
outliers_robust = set(df_imputed[key].index.difference(df_no_outliers['Robust covariance Estimator '+ key].index))

# Find the common outliers
common_outliers = outliers_isolation.intersection(outliers_lof)
print(f'Number of common outliers: {len(common_outliers)}')

common_outliers = outliers_isolation.intersection(outliers_robust)
print(f'Number of common outliers: {len(common_outliers)}')


# Find the common outliers
common_outliers = outliers_robust.intersection(outliers_lof)
print(f'Number of common outliers: {len(common_outliers)}')
```

```
Number of common outliers: 48
Number of common outliers: 41
Number of common outliers: 62
```

While all the outlier removers removed an equal amount:100, we know from above that the removed elements werent all the same. And the most common outliers taken out were by Robust and Local Outlier Factor

Having done Normalization combination over Imputation combination over Outlier Removal combination, all of them combine to make a huge dictionary of dataframes. We will iterate over the key of

dictionary and capture data frames and run the necessary conversions on our data to train multiple models in one go and see the results.

On default parameters, we observe the following results:

```
results_df.sort_values(by=['F1-score'],ascending=False)
```

	Normalization & Outlier & Imputer	Model	Accuracy_Train	Accuracy_test	Precision	Recall	F1-score	Confusion Matrix
125	Local Outlier Factor Robust_Median	XGBoost	0.963889	0.894444	0.879630	0.940594	0.909091	[[66, 13], [6, 95]]
95	Local Outlier Factor Robust_KNN	XGBoost	0.963889	0.894444	0.879630	0.940594	0.909091	[[66, 13], [6, 95]]
110	Local Outlier Factor Robust_Mean	XGBoost	0.963889	0.894444	0.879630	0.940594	0.909091	[[66, 13], [6, 95]]
55	Robust covariance Estimator Standard_KNN	XGBoost	0.963889	0.900000	0.871287	0.946237	0.907216	[[74, 13], [5, 88]]
10	Robust covariance Estimator MinMax_KNN	XGBoost	0.963889	0.900000	0.871287	0.946237	0.907216	[[74, 13], [5, 88]]
...	...	...	...	...	...	...	...	...
38	Local Outlier Factor MinMax_Median	Logistic Regression	0.737500	0.677778	0.673684	0.703297	0.688172	[[58, 31], [27, 64]]
1	Isolation Forest MinMax_KNN	SVM	0.787500	0.672222	0.711111	0.659794	0.684492	[[57, 26], [33, 64]]
16	Isolation Forest MinMax_Mean	SVM	0.787500	0.672222	0.711111	0.659794	0.684492	[[57, 26], [33, 64]]
31	Isolation Forest MinMax_Median	SVM	0.787500	0.672222	0.711111	0.659794	0.684492	[[57, 26], [33, 64]]
22	Local Outlier Factor MinMax_Mean	MLP	0.775000	0.672222	0.673913	0.681319	0.677596	[[59, 30], [29, 62]]

135 rows × 8 columns

There were 135 combinations in total and the following models were tried out; XGBoost (Boosting), SVM, MLPClassifier, Logistic Regression & Random Forest (Bagging).

We see that all combinations of XGBoost outperform all combinations of every other model that we picked. While we do see overfitting here, it is not the focus as the goal was to only observe what approach and model is performing best. We see Local Outlier Factor with Robust Normalization and Median, Mean, KNN Imputation doing the best in terms of F1-Score which weighs in Precision & Recall to assign a score and only ever reaches a perfect when both of them are a perfect 1.

Interesting Note:

It is clear that it was a good decision to keep the combinations alive and justify them with the data that speaks for itself. If all imputations were equal, then Robust Covariance Estimator Standard_KNN/Mean/Median would come together but it does not, instead, MinMax_KNN takes the next spot.

Afterwards, we use pipeline to make our code modular in nature alongside RandomSearchCV over GridSearchCV as it was much faster albeit how fast anything could be when they have to traverse over as many data frames as it had to in my case. In the end, we select the top 10 algorithms, which are all variants of XGBoost, and some are better in terms of Precision, some in Recall, and some have a better Train-Test Accuracy gap.

```
results__= results_df.copy(deep=True)
#results__ = results__.query("Accuracy_Train - Accuracy_test <= 0.05") #avoiding large gaps between Train-Test Accuracy first
results__ = results__.query("F1_score >= 0.90")
results__ = results__.sort_values(by=['F1_score','Precision','Recall'],ascending=[False,False,False])
results__ = results__.query("Model == 'XGBoost'")
results__
```

	Normalization & Outlier & Imputer	Model	Accuracy_Train	Accuracy_test	Precision	Recall	F1_score	Confusion Matrix	best_params	best_estimator
95	Local Outlier Factor Robust_KNN	XGBoost	1.000000	0.911111	0.882883	0.970297	0.924528	[[66, 13], [3, 98]]	{'model__n_estimators': 200, 'model__max_depth': 5, 'model__learning_rate': 0.05, 'model__gamma': 0.1, 'model__eta': 0.5}	(XGBClassifier(eta=0.5, gamma=0.1, learning_rate=0.05, max_depth=5,\n n_estimators=200))
110	Local Outlier Factor Robust_Mean	XGBoost	1.000000	0.905556	0.875000	0.970297	0.920188	[[65, 14], [3, 98]]	{'model__n_estimators': 100, 'model__max_depth': 10, 'model__learning_rate': 0.15, 'model__gamma': 0.0, 'model__eta': 0.5}	(XGBClassifier(eta=0.5, gamma=0.0, learning_rate=0.15, max_depth=10))
50	Local Outlier Factor Standard_KNN	XGBoost	1.000000	0.911111	0.896907	0.935484	0.915789	[[77, 10], [6, 87]]	{'model__n_estimators': 200, 'model__max_depth': 10, 'model__learning_rate': 0.3, 'model__gamma': 0.3, 'model__eta': 0.1}	(XGBClassifier(eta=0.1, gamma=0.3, learning_rate=0.3, max_depth=10,\n n_estimators=200))
65	Local Outlier Factor Standard_Mean	XGBoost	1.000000	0.911111	0.896907	0.935484	0.915789	[[77, 10], [6, 87]]	{'model__n_estimators': 50, 'model__max_depth': 10, 'model__learning_rate': 0.3, 'model__gamma': 0.3, 'model__eta': 1}	(XGBClassifier(eta=1, gamma=0.3, learning_rate=0.3, max_depth=10,\n n_estimators=50))
80	Local Outlier Factor Standard_Median	XGBoost	1.000000	0.911111	0.896907	0.935484	0.915789	[[77, 10], [6, 87]]	{'model__n_estimators': 50, 'model__max_depth': 10, 'model__learning_rate': 0.3, 'model__gamma': 0.3, 'model__eta': 1}	(XGBClassifier(eta=1, gamma=0.3, learning_rate=0.3, max_depth=10,\n n_estimators=50))

After multiple runs, sometimes Accuracy_Train would go 1, and most of the times we would succeed in curbing it to a value closer to what Accuracy_Test had, and in the end, eyeballing predictions per model and distribution of labels was the best way to understand thresholding is required.
Before we proceed further, it is important to provide a detailed response on selected evaluation criteria.

Evaluation Criterion and Prediction

F1 Score is the perfect balance in a way; we get to look at both Precision and Recall and only ever reach a perfect score of 1 if both precision and recall are 1.

- Recall technically is about all anomalies being identified correctly, whereas Precision as the name suggests is about counting only anomalies, in this case whether the identification of difficulty with paying electricity bill was done correctly classified and how many times was that the case (True Positive/True+False (Positives))

- Simply Said; High precision means that the model is good at avoiding false positives, and High Recall means that the model is good at detecting all the positive instances.

To note, knowing that if the cost of false negatives is higher, then recall should be used as an evaluation metric, and if the cost of false positives is higher, then precision should be used as an evaluation metric, and with AENERGY scenario, we would prefer Precision as we do not want to say a person can pay when they can't, maybe they will lose out on certain eligible cuts or benefits given their condition.

My choice of evaluation metric for this is F1-Score, then Precision, and then Recall.

Given we know that the dataset was balanced, we should expect the distribution shift to not be so large on the prediction time either, which was occurring in some places with our model.
To solve, I used a Threshold Criteria:

```
[ ] for column in predictions_df.columns:
        print(f"Value counts for column {column}: {predictions_df[column].value_counts()}")
```

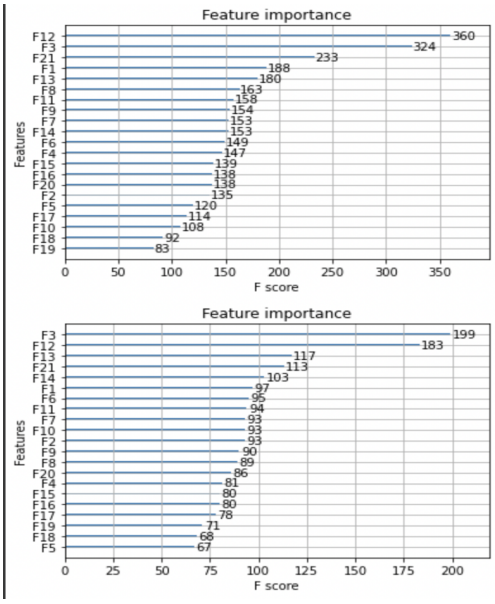
```
1    569
0    431
Name: prediction_1, dtype: int64
```

To remove predictors that deviate far too much from the reality that was fed, we remove such predictors that behave abnormally.

Thresholding Predictors

```
# Threshold for the value_counts difference
threshold = 250

# Get the value_counts of each column in predictions_df
value_counts = predictions_df.apply(lambda x: x.value_counts())

# Calculate the difference between the value_counts of each column
difference = value_counts.iloc[0] - value_counts.iloc[1]

# Get the column names where the difference is greater than the threshold
to_drop = difference[difference > threshold].index

# Drop the columns from predictions_df
predictions_df = predictions_df.drop(columns=to_drop)
```

Finally, feature importance for two top variants of XGBoost Classifier:



We can see F12, F21 and F3, F13 topping the charts majorly, which is a good thing because we already know how important F21 was from earlier correlation charting. F12 and F3 were also strongly positively correlated which also appears to be the case in terms of feature importance for our models.

I end up collecting predictions across columns of a dataframe and conduct a majority vote and base the final column to be part of the prediction_csv that is then attached for AENERGY.

Conclusion

It is good to see XGBoost perform so healthy and actively. It is one of the best models out there for tabular datasets, and having used it in professional capacity over terabytes of data and see it be competitive on all kinds of dataset sizes, on majority of classification problems that are phrased correctly, this decision tree form of boosting is one of the best out there. It solves AENERGY's problem statement almost as accurately as they need it to be, and the voting structure can easily allow for flexibility in terms of the desire to improve Precision or Recall per customization, and as of the time of this report, it is currently optimized for Precision.

AENERGY's task for predicting customer's annual expenditure variation

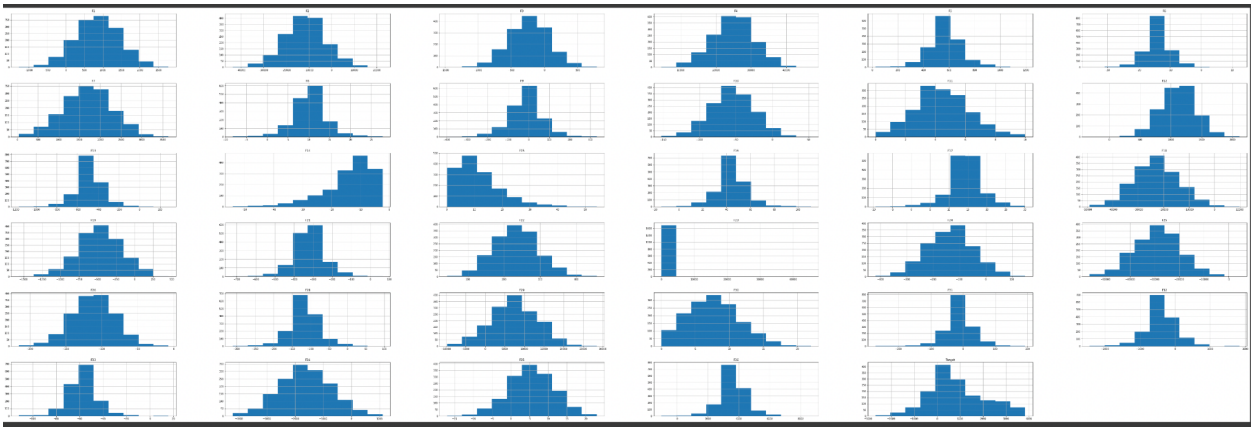
Regression Task

Problem Statement

The goal is to design a system to predict the variation in the annual expenditure that a customer is going to have due to the increase of the energy cost (in £/year, positive if next year the customer is going to spend more, negative if they are going to spend less). This means our target variable will have both positive and negative values.

Data Understanding

First off, we have no null values in this dataset. The dataset collected by AENERGY for this task captures about 36 features, and one target variable. Two of these features are categorical and need to be encoded appropriately.



We have a distribution for features and we see more features in a normal distribution than the last task.

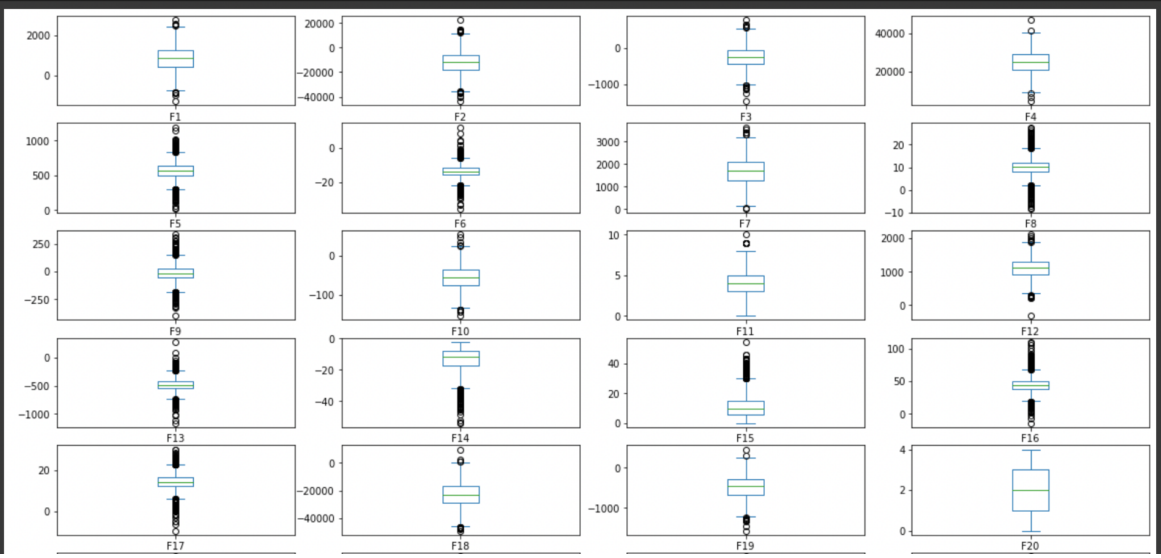
Feature Engineering & More:

Before we move further, we need to convert our features that are categorical.
F20 has Very Low, Low, Medium, High, Very High which is encoded in 0,1,2,3,4 as it shows increase and should capture that 0<1<2<3<4.
For F27, I have made use of dummy and converted it into a one-hot encoded feature instead.

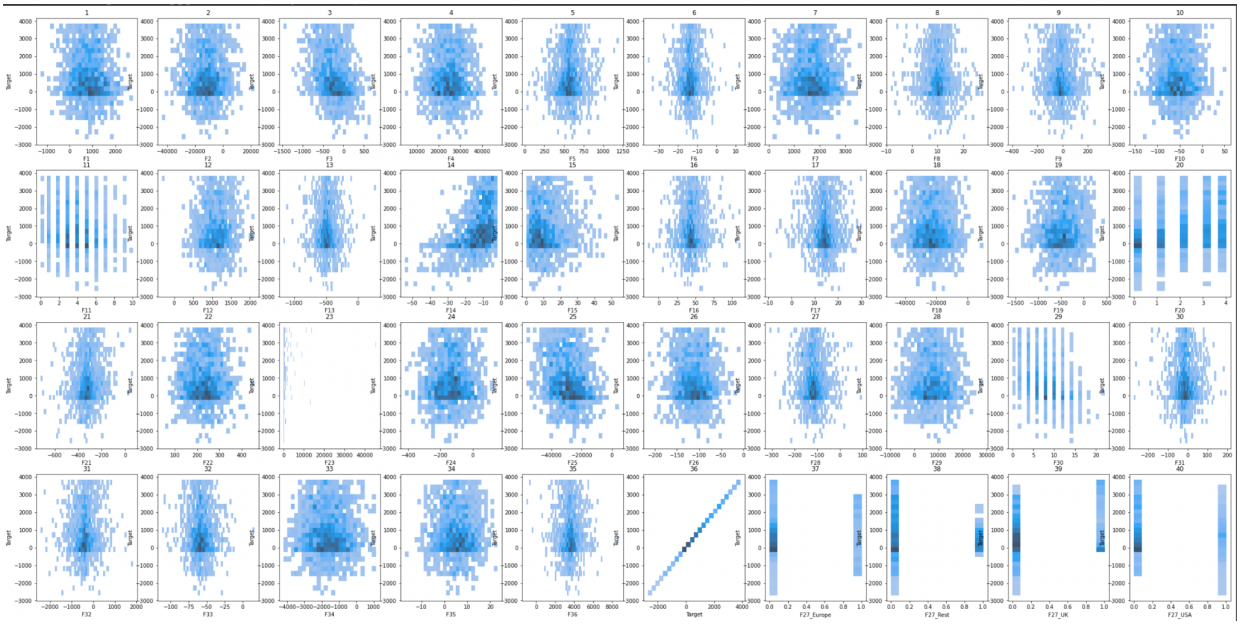
We take a peek using Box Plot for outlier eyeballing, and notice their presence in some of our features.

We see noticeable presence of outliers in a lot of features like F11, F23, F25, but it would be good to visualize this again after normalization. We already could see this coming after we described the dataframe above with means and max moving in polar opposite directions for some of the features like F2 (Mean: -11K, Max: 22K)

```
AENERGY.plot(kind='box', subplots=True, figsize=(20, 40),layout=(20,4))
plt.show()
```

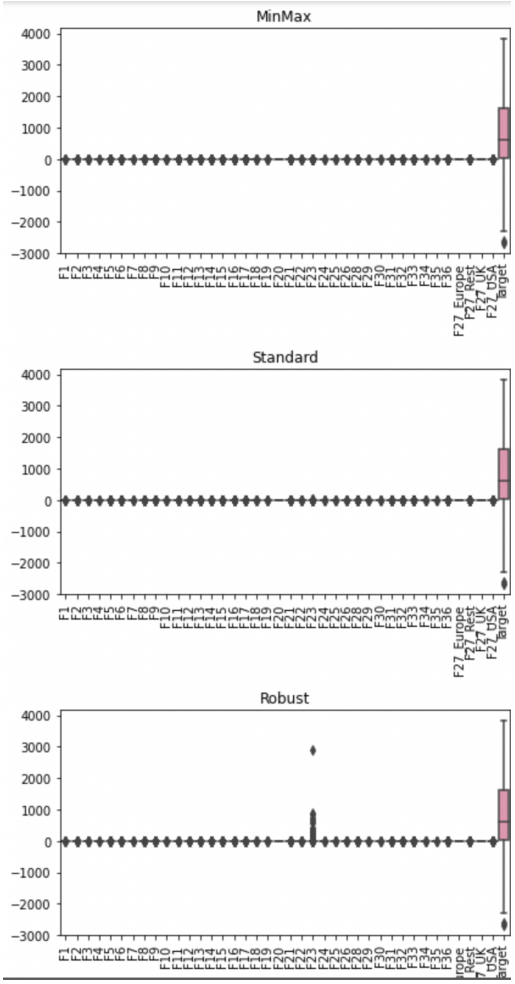


Next, we visualize every feature against the target variable to see their influence on each other.



After our initial understanding of the dataset is done, we can dissect the dataset further with the same systematic procedures followed previously; applying outlier removal & normalization.
This time however, we will utilize both the understanding gained from the previous experiment and further enhance our code to keep together the necessary bits.

For example, in our attempt to normalize, we see a similar chart for MinMax, and we choose to not move forward here. Instead, we utilize more models over more data variants to see if we can get a variety of algorithms to pool in for an aggregated output in the end.



After applying Standard and Robust on our dictionary of dataframes, we move forward with outlier methods; Local Outlier Factor which was a clear winner in the last task, and fit it on a default Linear Regression and gauge RMSE.

The first result is with Local Outlier Factor; Standard, and the second one is with Robust.

```
rmse_train: 0.513
rmse_test: 0.564


rmse_train: 0.388
rmse_test: 0.401
```

We then try out another model to confirm our confluence here. We see that indeed with Robust our numbers look better already.

```
for key in df_no_outliers:
  X_train, X_test, y_train, y_test = train_test_split(df_no_outliers[key].drop(columns="Target"), df_no_outliers[key]['Target'], test_size=0.3)
  model_regression=XGBRegressor()
  model_regression.fit(X_train,y_train)
  print("Normalization In Effect: ",key)
  y_predtr_model=model_regression.predict(X_train)
  y_pred_model=model_regression.predict(X_test)
  print("rmse_train:",round(np.sqrt(mean_squared_error(y_train, y_predtr_model)),3))
  print("rmse_test:",round(np.sqrt(mean_squared_error(y_test, y_pred_model)),3))
  print("\n")
```

```
[23:05:39] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Normalization In Effect:  Local Outlier Factor Standard
rmse_train: 0.226
rmse_test: 0.435


[23:05:39] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Normalization In Effect:  Local Outlier Factor Robust
rmse_train: 0.179
rmse_test: 0.327
```

Model Training & Evaluation

Multiple Models Training -> Finetune Approach

```
[ ] def run_random_search(regressor, parameters, X_train, y_train):
        pipeline = Pipeline([('regressor', regressor)])
        random_search = RandomizedSearchCV(pipeline, param_distributions=parameters, n_iter=10, cv=5, n_jobs=-1)
        random_search.fit(X_train, y_train)
        return random_search
```

We build the following function and make a parameters and regressor list below this code piece to feed iteratively and store the results and save the models simultaneously.

Finally, we are welcomed by the following results.

	Normalization & Outlier & Imputer	Model	RMSE_train	RMSE_test	best_params
0	Local Outlier Factor Standard	Linear Regression	642.3189	638.6553	(LinearRegression())
1	Local Outlier Factor Robust	Linear Regression	620.9489	612.9126	(LinearRegression())
2	Local Outlier Factor Standard	Lasso	637.1253	650.8020	(Lasso())
3	Local Outlier Factor Robust	Lasso	617.2134	621.8430	(Lasso())
4	Local Outlier Factor Standard	Ridge	641.9079	639.6084	(Ridge())
5	Local Outlier Factor Robust	Ridge	604.2631	650.6736	(Ridge())
6	Local Outlier Factor Standard	Gaussian Process Regressor	0.0000	0.0000	(GaussianProcessRegressor(normalize_y=True))
7	Local Outlier Factor Robust	Gaussian Process Regressor	0.0000	0.0000	(GaussianProcessRegressor(normalize_y=True))
8	Local Outlier Factor Standard	Support Vector Regressor	1237.5903	1186.3401	(SVR())
9	Local Outlier Factor Robust	Support Vector Regressor	1201.7318	1218.3027	(SVR())
10	Local Outlier Factor Standard	XGBRegressor	240.3945	243.0684	(XGBRegressor(learning_rate=0.1112000000000000...
11	Local Outlier Factor Robust	XGBRegressor	233.8540	246.9024	(XGBRegressor(learning_rate=0.1112000000000000...
12	Local Outlier Factor Standard	RandomForestRegressor	275.2168	294.2300	((DecisionTreeRegressor(max_features='auto', m...
13	Local Outlier Factor Robust	RandomForestRegressor	348.0879	341.3066	((DecisionTreeRegressor(max_features='auto', m...
14	Local Outlier Factor Standard	MLPRegressor	511.0769	502.5983	(MLPRegressor(max_iter=800, power_t=0.8, warm_...
15	Local Outlier Factor Robust	MLPRegressor	177.8329	162.8876	(MLPRegressor(power_t=0.8, solver='lbfgs', val...
16	Local Outlier Factor Standard	ElasticNet	636.3684	652.9746	(ElasticNet(alpha=2, copy_X=False, l1_ratio=1,...
17	Local Outlier Factor Robust	ElasticNet	626.1109	601.3603	(ElasticNet(alpha=2, fit_intercept=False, l1_r...

Interesting! Gaussian Process Regressor has a 0 RMSE.

Let us deep dive into the kind of predictions it is making on the test_dataset to see whether values are changing or not, there is a probability that posterior covariance between each of my training examples and each test example is 0 or very close to 0, so the model just predicts the mean function for each test example.

After carrying out a small experiment with Gaussian Process Regressor, we choose to discard it due to the wild variations that came with its results, I ran it outside of this code loop to see whether the cross validated score would once again appear 0, and then extend on to it by manually viewing scores it gave on the test data and compared it with another randomly selected model (XGBRegressor) to see whether the predictions differ. They differed to the point where Gaussian Process predictions were scientifically noted.

```
ok.head(15)
```

	0
0	-6.359765e-07
1	-2.620792e-06
2	-3.858905e-05
3	-5.674098e-05
4	2.847587e-05
5	-9.839349e-08
6	1.727911e-06
7	7.018057e-05
8	-3.694612e-05
9	-5.047880e-06
10	1.338675e-08
11	-2.840233e-06
12	0.000000e+00
13	-3.344116e-05
14	5.724225e-06

Interesting, the results are varying also. But they seem really small. To not risk the skewness, I will drop this model and use the rest - best 3 for averaging their results and placing them in P3_Test for AENERGY's expenditure goal.

Evaluation Criterion:

Based on the task, we choose RMSE which is an abbreviation of Root Mean Squared Error. The lower or closer to 0 the better it is. If we normalized the column of target it would appear much more differently (It was initially tried but then I had a hard time inverse transforming it so I left the target column untouched)

Root Mean Squared Error,Mean Squared Error, Mean Absolute Error are common metrics that are used for regression tasks, and we select RMSE because that is what would be used to gauge performance over in the AENERGY organization for which we curate this task.

The best performer here was MLPRegressor with Local Outlier Factor and Robust Outlier Removal.

	Normalization & Outlier & Imputer	Model	RMSE_train	RMSE_test
15	Local Outlier Factor Robust	MLPRegressor	177.8329	162.8876
10	Local Outlier Factor Standard	XGBRegressor	240.3945	243.0684
11	Local Outlier Factor Robust	XGBRegressor	233.8540	246.9024
12	Local Outlier Factor Standard	RandomForestRegressor	275.2168	294.2300

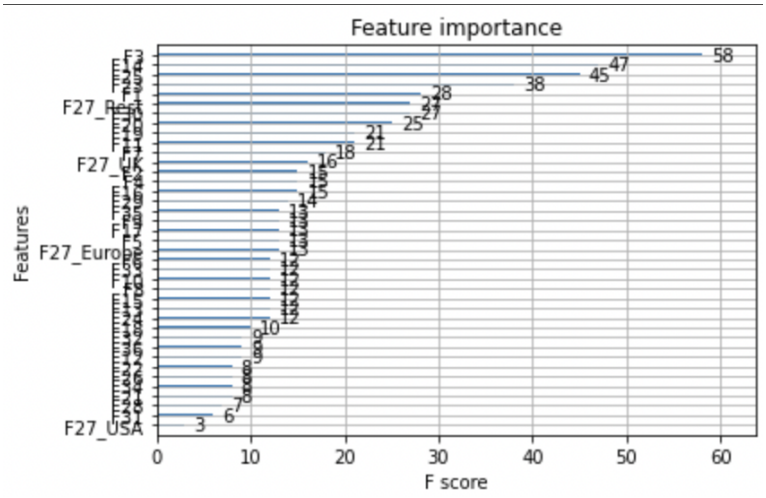
Prediction:

For prediction, I increased the weights on better algorithms by repeating their column;

	prediction_0	prediction_1	prediction_2	0_1	2_1	2_2	Target
0	187.237477	421.278717	-233.870941	187.237477	-233.870941	-233.870941	15.690141
1	-967.230174	-344.513184	-846.192871	-967.230174	-846.192871	-846.192871	-802.925358
2	209.117469	726.249268	-275.622864	209.117469	-275.622864	-275.622864	52.935936
3	-847.228894	289.467804	-942.142151	-847.228894	-942.142151	-942.142151	-705.236073
4	3203.928881	2590.018555	3021.639893	3203.928881	3021.639893	3021.639893	3010.465999

This simple and manual correction allows for more weightage to the better algorithms in the list.
Note: 0_1 implies 1st replica of prediction_0.

We also draw out feature importance to see F3 and F14 top the chart with respect to XGBRegressor.



Conclusion:

It was a surprise to see MLPRegressor outperform other options in the list by a margin. But given the solver used to solve it was lbfgs which is notorious for performing extraordinarily well for smaller datasets and does not utilize stochastic gradient descent for which MLPs are known, it makes total sense. To conclude, we average the results i.e ensemble to give the final prediction with manual weighing through multiplying columns.