NSSC DATA ANALYTICS

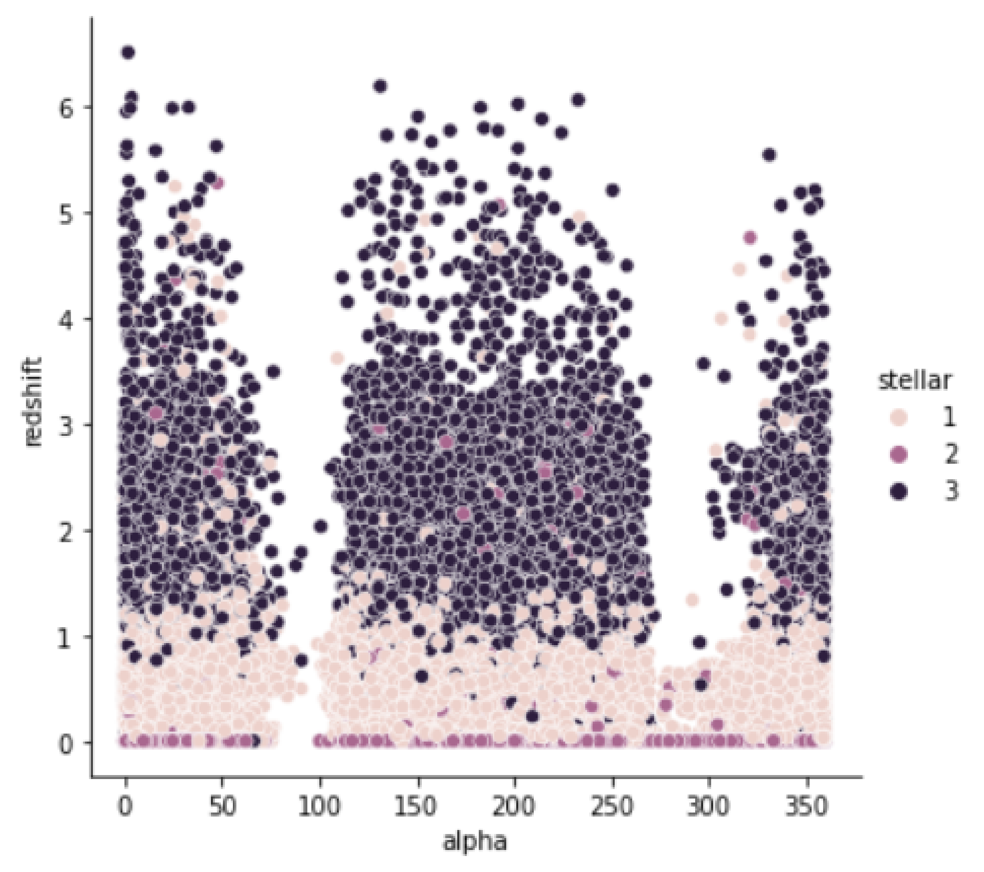
TEAM LEFT AND RIGHT

DATA EXPLORATION

```
train.describe()
```

	id	ultraviolet_filter	green_filter	red_filter	near_infrared_filter	alpha	delta	redshift
count	134911.000000	134911.000000	134911.000000	134911.00000	134911.000000	134911.000000	134911.000000	134911.000000
mean	67456.000000	21.954199	20.620361	19.52772	18.926294	176.110655	25.061154	0.473703
std	38945.595421	2.357727	2.278216	2.08472	1.982872	91.469418	19.055886	0.621603
min	1.000000	14.381980	8.645090	10.69846	10.477428	0.014814	-14.272665	-0.001592
25%	33728.500000	20.009710	18.799620	17.73831	17.322310	136.827729	7.018153	0.027116
50%	67456.000000	22.284200	21.168940	20.12401	19.275063	175.356844	25.857613	0.352549
75%	101183.500000	23.651765	22.373330	21.03392	20.316915	219.337308	39.903438	0.596258
max	134911.000000	27.842590	28.035390	26.89342	27.153450	359.875028	73.112284	6.500708

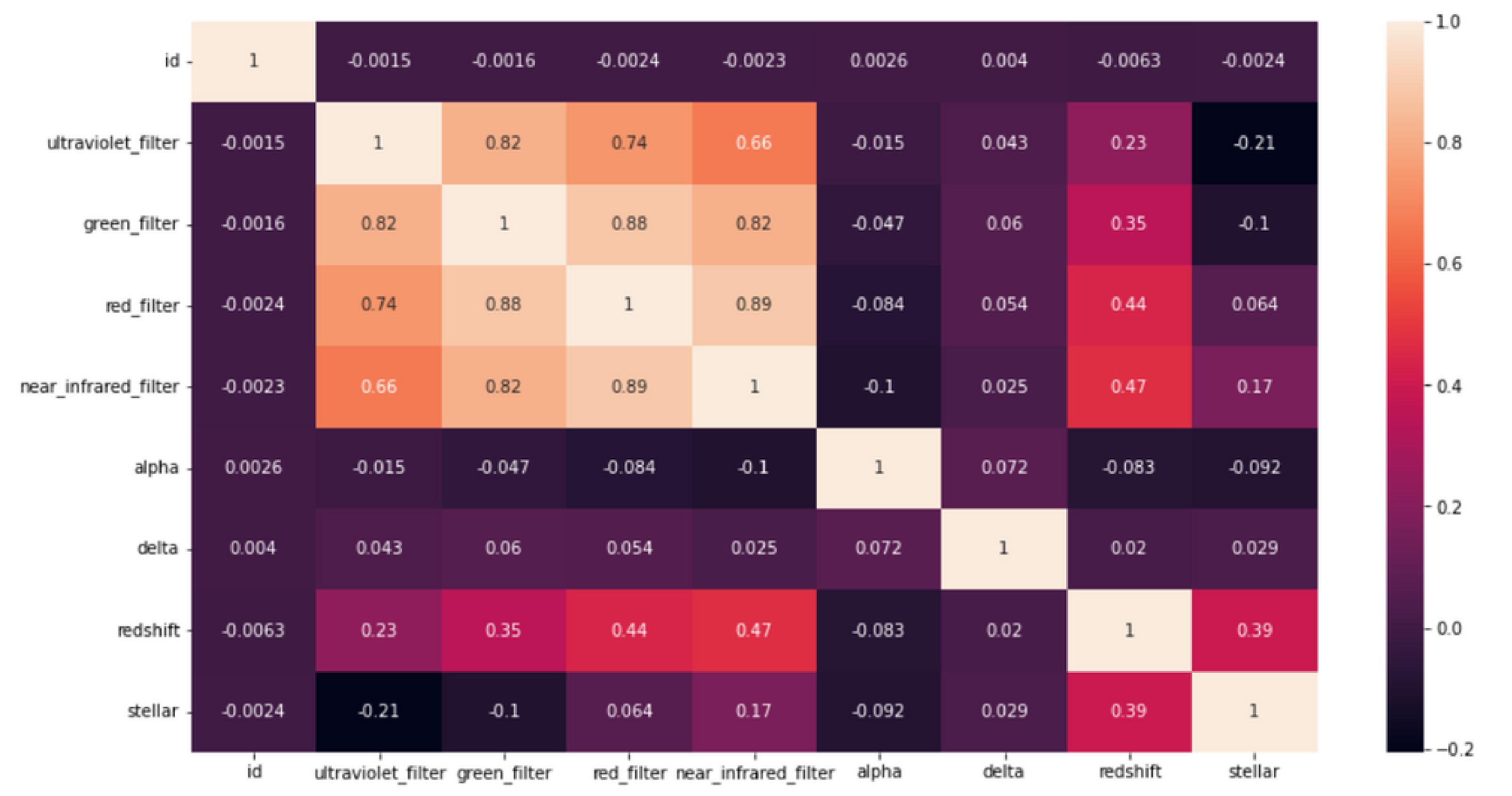
<seaborn.axisgrid.FacetGrid at 0x2e13a432be0>



- *There were no categorical values, only numerical features. Datatypes were all integers or floats.*
- *Checked for null values, there were no null values.*
- Looked at the counts of stellar with 1 comprising the dataframe 86701 (~64.2% of the dataset), so the dataset was slightly imbalanced but we didn't feel the need to upsample/downsample data.
- Plotted a relplot between redshift and alpha and noticed that stellar was '3' for higher values of redshift (probably outliers),
- *Dropped the id column since it wasn't meaningful to keep it for training.*

HEATMAP & COLUMN FEATURES

```python
plt.figure(figsize = (15,8))
ax = sns.heatmap(train.corr(), annot=True)
```



Plotting

Imported the seaborn library and plotted a correlation matrix, using seaborn helped us spot highly correlated features easily. The lightly coloured boxes indicate high correlation.

Interpretation

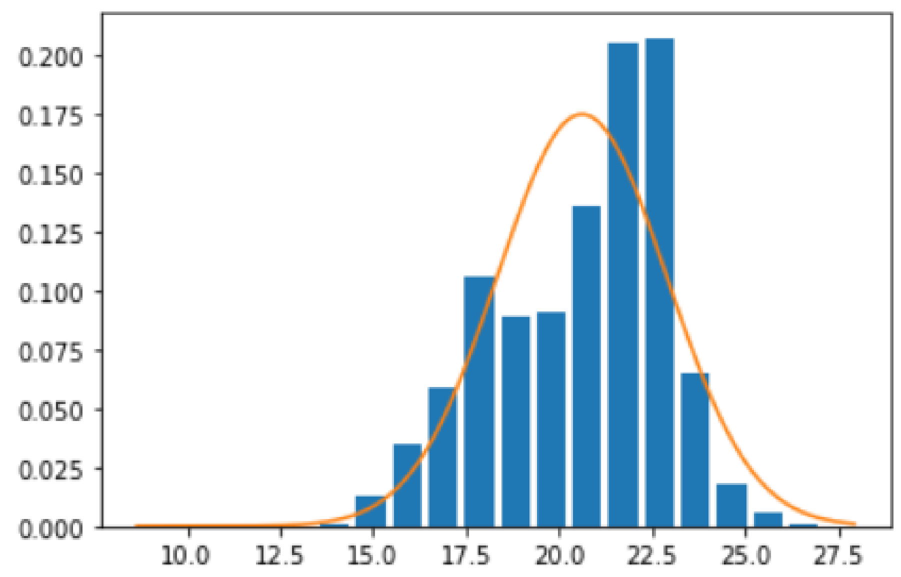
'green_filter', 'red_filter' and 'near_infrared_filter' had very high correlation with each other. This denotes that the features aren't completely independent of each other.

Checking for distribution

Plotted the correlated columns one by one to check which one has normal distribution, ML models learn easily if data is normally distributed.

```python
plt.hist(train['green_filter'], bins = 20, rwidth = 0.8, density = True)

rng = np.arange(train.green_filter.min(), train.green_filter.max(), 0.1)
plt.plot(rng, norm.pdf(rng, train.green_filter.mean(),train.green_filter.std()))
```

```
[<matplotlib.lines.Line2D at 0x186c9447250>]
```



Defined a function

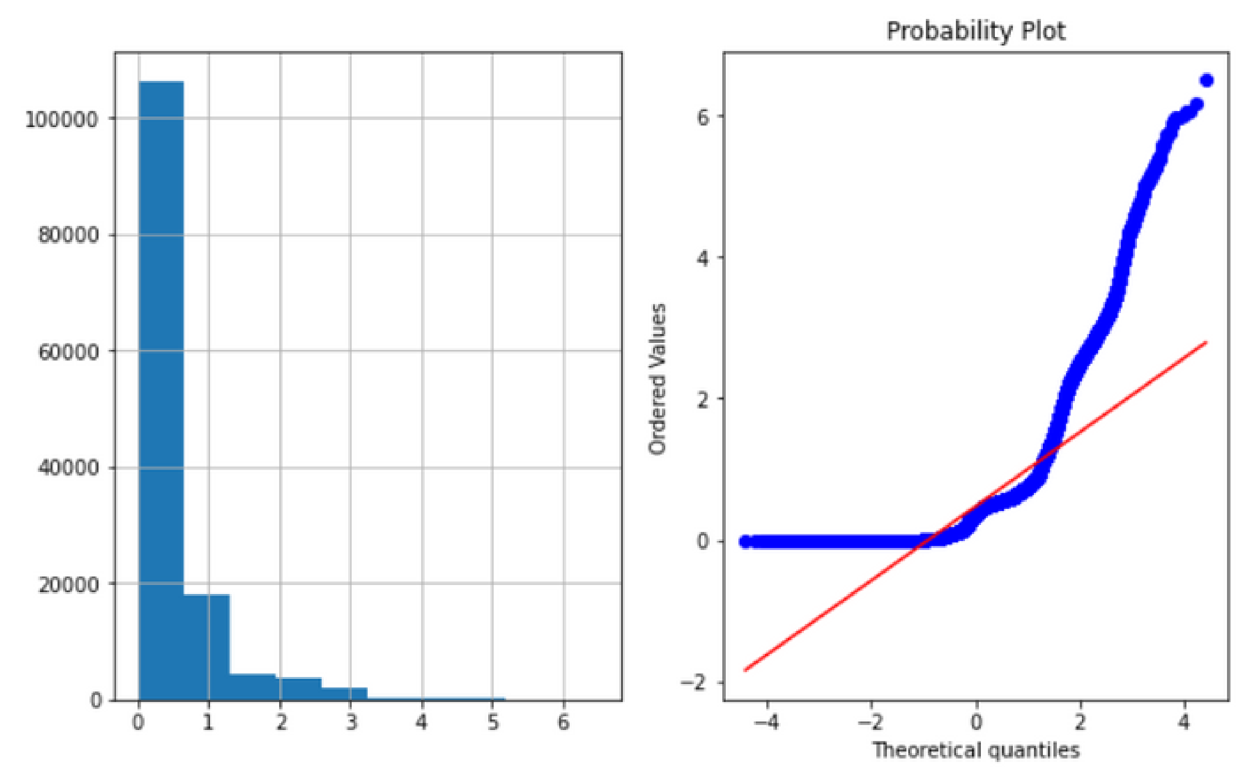
The code was cumbersome so we defined a function with the column plot and probability plot (known as QQ plot) which would help us visualise better.

COLUMN FEATURES

Code-

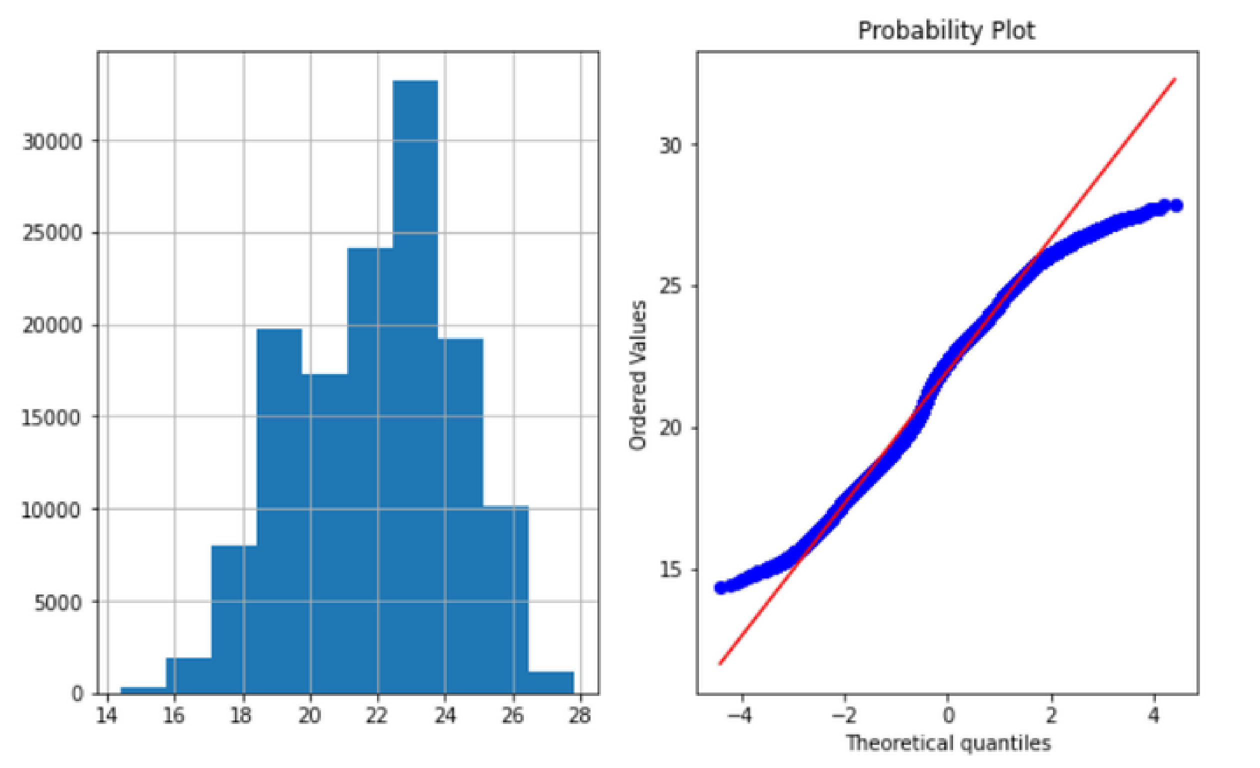
```python
# this is the function
def plot_data(df, feature):
    plt.figure(figsize = (10, 6))
    plt.subplot(1,2,1)
    df[feature].hist()
    plt.subplot(1,2,2)
    stats.probplot(df[feature], dist = 'norm', plot=pylab)
    plt.show()
```

```python
plot_data(train, 'redshift')
```



Output-

```python
plot_data(train, 'ultraviolet_filter')
```



- The ultraviolet filter column is almost normally distributed which is a good sign.
- *The redshift column is rightly skewed and it also has quite a few outliers*
- SImilar plots were done for all columns and the .skew() function was called to check the skewness of the data.

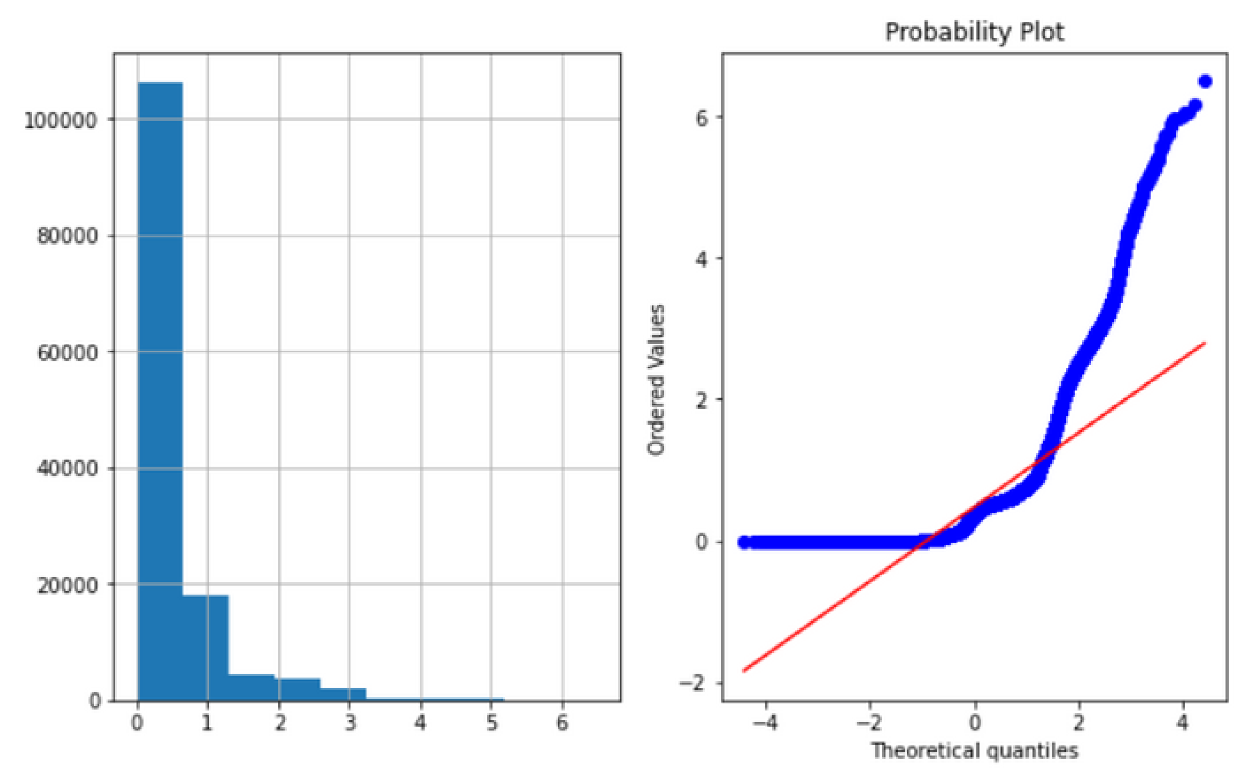
OTHER DATA EXPLORATION

- *Used mutual info gain to check the importance of each column*

Code-

```
# train.shape
mutual_info = mutual_info_classif(train, y)
mutual_info= pd.Series(mutual_info)
mutual_info.index = train.columns
```

```
plot_data(train, 'redshift')
```



- *Output-*

mutual_info

id	0.000000
ultraviolet_filter	0.109334
green_filter	0.097419
red_filter	0.058408
near_infrared_filter	0.072340
alpha	0.054370
delta	0.086306
redshift	0.631598
stellar	0.887168

dtype: float64

- *Redshift gave the highest information gain after stellar (which was the target class)*
- This means that redshift was most important and it had to be transformed to fit a normal dsitrbution.
- SImilar plots were done for all columns and the .skew() function was called to check the skewness of the data.

HANDLING OUTLIERS

- Defined a function to detect outliers using IQR (Interquartile Range)

Code-

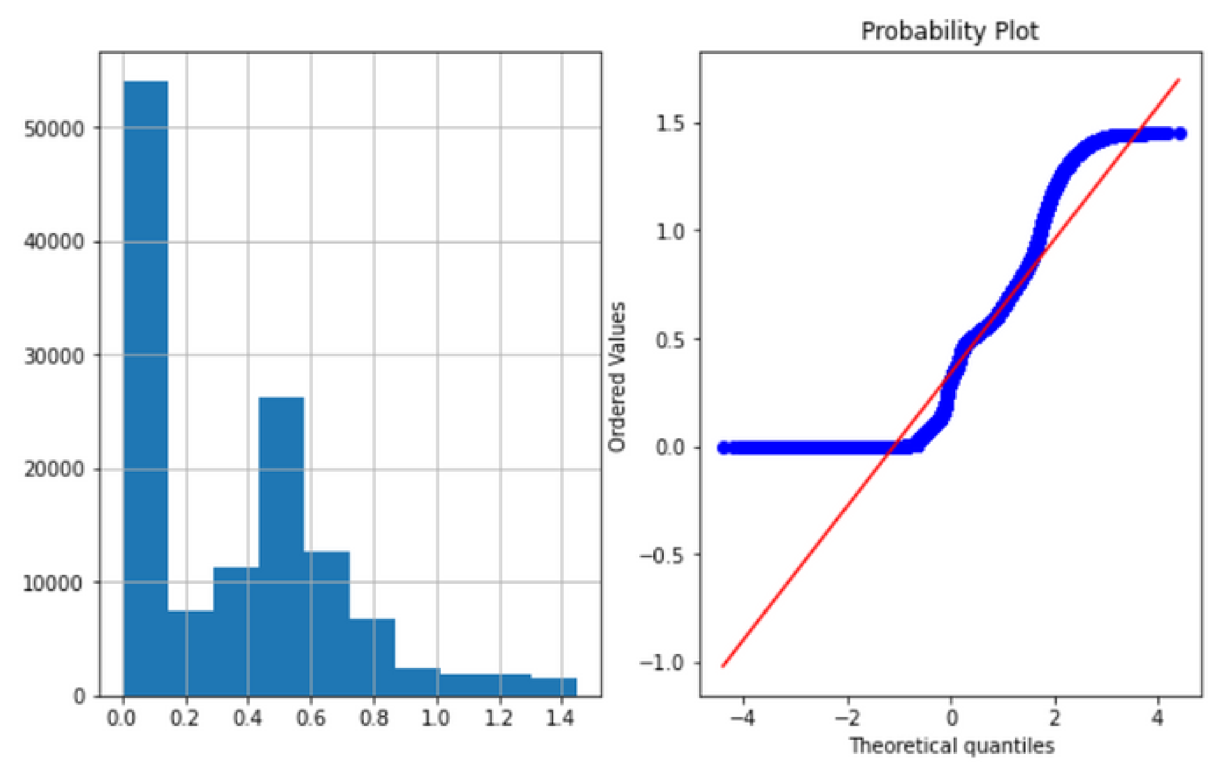
```
def  outliers(df, feature):
    IQR = df[feature].quantile(0.75) - df[feature].quantile(0.25)

    lower_range = df[feature].quantile(0.25) - (IQR*1.5)
    upper_range = df[feature].quantile(0.75) + (IQR*1.5)

    print(lower_range, upper_range)
```

- *Output-*

```
outliers(train, 'redshift')
```

```
-0.8265965047438749 1.4499700459456585
```

```
plot_data(train_1, 'redshift')
```



- *ReFOund the upper and lower range for detecting outliers*

- *The plot of the redshift column (to the left) is without outliers, the skewness has clearly reduced.*

- The same was going to be applied to alpha but we would have lost a huge chunk of our dataset, so outlier removal was done only with respect to redshiift.

- *The outliers were capped earlier but that reduced the score so they were removed completely.*

MODELS USED (F1 SCORE)

ADABOOST

```
f1_score(y_test, y_pred, average='macro')
0.90105010664835
```

XGBCLASSIFIER

```
f1_score(y_test, y_pred, average='macro')
0.9086286128027368
```

RANDOM FOREST

```
f1_score(y_test, y_pred, average='macro')
0.906761054902089
```

Although gradient boosting algorithms are said to not be sensitive to outliers but it can be bad for boosting because boosting builds each tree on previous trees' residuals/errors.

We tried removing outliers (with respect to 'redshift') which led to a better score although we lost ~20% of the dataset.

XGBClassifier was finalised due to better f1 score.

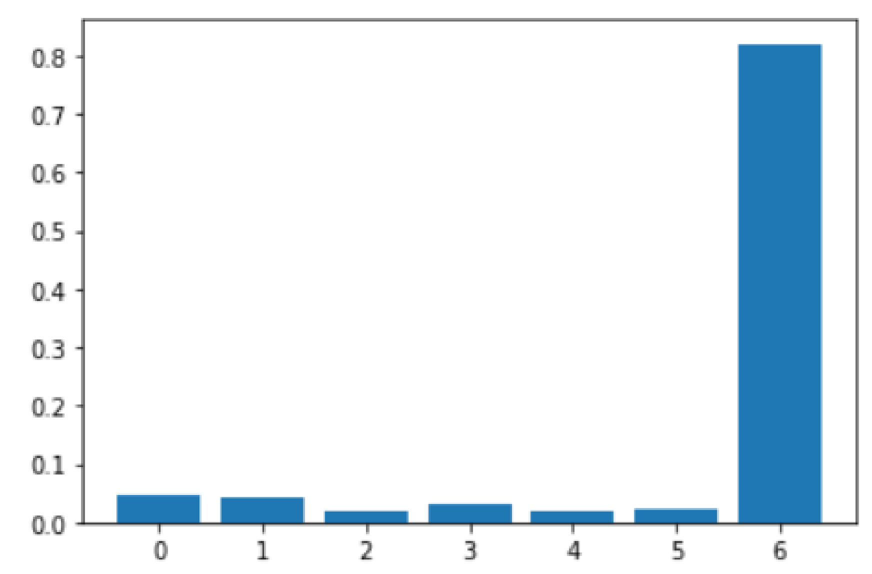
HYPERPARAMETER TUNING

- *Did hyperparameter tuning for XGBClassifier since that gave the best score.*

Code-

```
xgbclf = XGBClassifier(objective="multi:softmax", tree_method='hist')
clf = RandomizedSearchCV(estimator=xgbclf,
                         param_distributions=params,
                         scoring='accuracy',
                         n_iter=25,
                         n_jobs=4,
                         verbose=1)
clf.fit(train_1, y)

best_combination = clf.best_params_
```

```
# plot
plt.bar(range(len(xgb.feature_importances_)), xgb.feature_importances_)
plt.show()
```



- *Output-*

```
best_combination
```

```
{'subsample': 0.6,
 'num_class': 10,
 'n_estimators': 750,
 'max_depth': 6,
 'learning_rate': 0.01,
 'colsample_bytree': 0.7,
 'colsample_bylevel': 0.7}
```

- These were the most important feature with respect to the model
- *Defined a range of values for each hyperparameter and applied randomizedSearchCV*
- The graph (to the left) shows the importance given by the model to columns.
- Redshift has been given very high importance as predicted.

FAILED ATTEMPTS

T

↓

TRANSFORM-ATIONS

Tried transforming the alpha and redshift columns using boxcox transformation and yeo-johnson (for redshift) but the distribution didn't change much.

N

↓

NEURAL NETWORK

Tried making a neural network with multiple layers and softmax activation. Didn't give a score close to other classifiers.

S

↓

SCALERS

Scalers didn't seem to work for the data.
Scalers tried - RobustScaler, MinMaxscaler, StandardScaler.

D

↓

DROPPING COLUMNS

Dropped columns like green filter, red filter, near infrared filter (applied PnC), the highly correlated columns but that didn't increase the score. So, kept all columns.

THANK YOU