Enrollment Forecast

Description of Dataset:

	Id	Fiscal Year	Term Type	Career	Program Level	Study Year	Campus_Id	Faculty Group	Program Grouping	Coop Regular	WorkTerm	Attendance	Visa Status	Gender	Unique Headcount
0	0	2008/09	Fall term	Graduate	Doctoral	D	4	AHS	Aging, Health and Well-Being	Regular	Academic Term	Full-Time	Canadian	Male	5
1	1	2008/09	Fall term	Graduate	Doctoral	D	4	AHS	Aging, Health and Well-Being	Regular	Academic Term	Full-Time	Canadian	Female	7
2	2	2008/09	Fall term	Graduate	Doctoral	D	4	AHS	Aging, Health and Well-Being	Regular	Academic Term	Full-Time	International	Female	1
3	3	2008/09	Fall term	Graduate	Doctoral	D	4	AHS	Health Studies and Gerontology	Regular	Academic Term	Full-Time	Canadian	Male	3
4	4	2008/09	Fall term	Graduate	Doctoral	D	4	AHS	Health Studies and Gerontology	Regular	Academic Term	Full-Time	Canadian	Female	11

We have a dataset which specifies parameters like the details of the program - the term, the faculty group, the study year, work term, and also the individual's details like nationality and gender.

A unique headcount is provided for several combinations of these parameters, so you'll be able to know that there are 5 female Canadian full-time regular doctoral students who are part of the AHS group in the AGWB program for the fall term of 2008.

Our job is to make a model to analyse these parameters and predict the headcount for a given set of parameters in future years.

Processing of Categorical Data:

	Id	Fiscal Year	Spring term	Winter term	Undergraduate	Doctoral	Masters	Non-Degree	Qualifying
0	72249	1516	0	0	0	1	0	0	0
1	72250	1516	0	0	0	1	0	0	0
2	72251	1516	0	0	0	1	0	0	0
3	72252	1516	0	0	0	1	0	0	0
4	72253	1516	0	0	0	1	0	0	0

pg_0	pg_1	pg_2	pg_3	pg_4	pg_5	pg_6	pg_7
2.0	0.0	-1.0	-1.0	0.0	-3.0	0.0	3.0
2.0	0.0	-1.0	-1.0	0.0	-3.0	0.0	3.0
2.0	0.0	-1.0	-1.0	0.0	-3.0	0.0	3.0
2.0	0.0	-1.0	-1.0	0.0	-3.0	0.0	3.0
2.0	0.0	-1.0	-1.0	0.0	-3.0	0.0	3.0

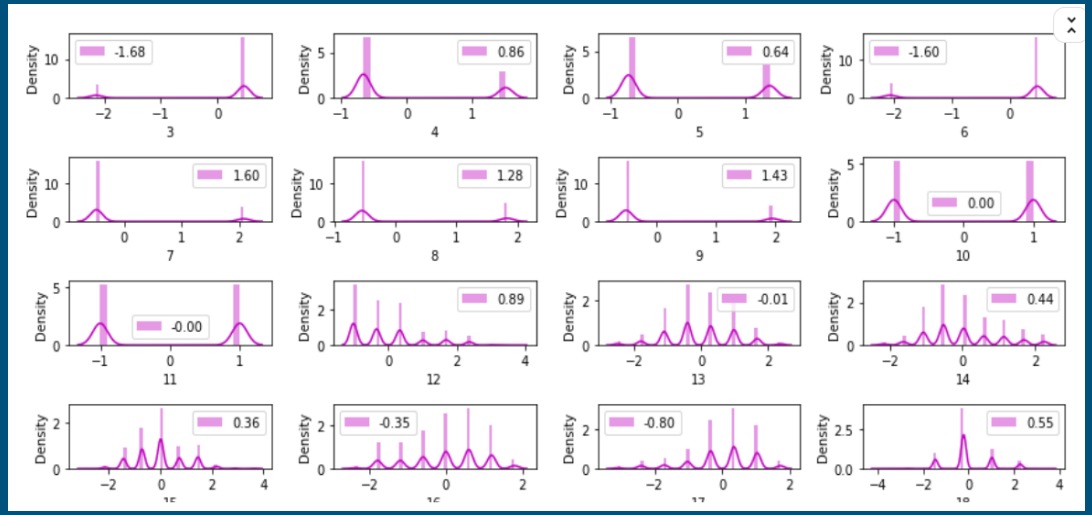
One-Hot Encoding:

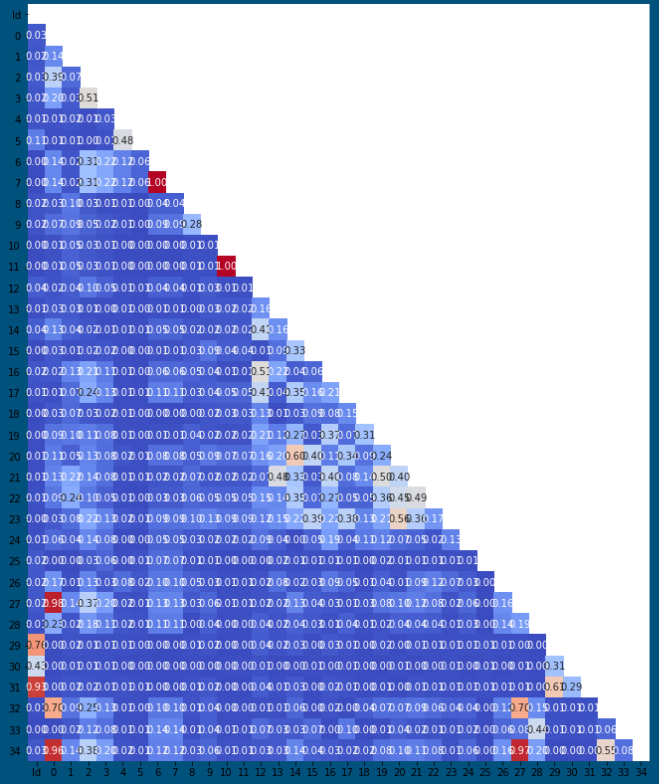
A one hot encoding allows the representation of categorical data to be more expressive. The categories must be converted into numbers, it takes a column which has categorical data, which has been label encoded, and then splits the column into multiple columns. The numbers are replaced by 1s and 0s, depending on which column has what value.

Feature hashing:

A fast and space-efficient way of vectorizing features, i.e. turning arbitrary features into indices in a vector or matrix. We can uniquely represent 147 features to 8 features.

Analysis of Dataset:





- The distplot shows the distribution of the data values of various features.
- The heatmap shows the correlation of each of the features which enables better selection of features to increase accuracy.

XGBoost Algorithm:

The XGBoost (eXtreme Gradient Boosting) is a supervised learning algorithm that accurately predicts a target variable by combining an ensemble of estimates from a set of more superficial or weaker models.

Its key principle, ensemble learning combines the predictive power of multiple learners resulting in a single model which gives the aggregated output from several models.

XGBoost uses boosting, where a number of trees are built sequentially such that each subsequent tree aims to reduce the errors of the previous tree.

The major advantages of XGBoost are its parallel learning capacities, ability to handle sparse data and a weighted quantile sketch algorithm to handle weighted data.

```
# Instantiate model with 100 decision trees
from xgboost.sklearn import XGBClassifier
from xgboost.sklearn import XGBRegressor
clf = XGBRegressor(random_state = 42,learning_rate=0.5, max_depth=8, n_estimators=120)

clf.fit(x_train, y_train)
```

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.5, max_delta_step=0, max_depth=8,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=120, n_jobs=4, num_parallel_tree=1, random_state=42,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

```
from sklearn.metrics import mean_squared_error, r2_score
import math
# Use the forest's predict method on the test data
predictions = clf.predict(x_test)
predictions = abs(predictions)
predictions = np.round(predictions)

# model evaluation
rmse = np.sqrt(mean_squared_error(y_test, predictions))
r2 = r2_score(y_test, predictions)
print('Root mean squared error: ', rmse)
print('R2 score: ', r2)
```

```
Root mean squared error:  6.552468174207496
R2 score:  0.8604825502422389
```

Optimization of XGBoost Algorithm:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=
42,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.5, max_delta_step=0, max_depth=8,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=120, n_jobs=4, num_parallel_tree=1, random_state=
42,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

Methods of Optimisation: Bayesian Optimisation, OPTUNA Hyperparameter Tuning, Randomised Grid Search Algorithm

```
# model evaluation
rmse = np.sqrt(mean_squared_error(y_test, predictions))
r2 = r2_score(y_test, predictions)
print('Root mean squared error: ', rmse)
print('R2 score: ', r2)
```

```
<class 'numpy.ndarray'>
Mean Absolute Error: 4.58 degrees.
Root mean squared error:  9.490243302027801
R2 score:  0.7073332519528963
```

```
# model evaluation
rmse = np.sqrt(mean_squared_error(y_test, predictions))
r2 = r2_score(y_test, predictions)
print('Root mean squared error: ', rmse)
print('R2 score: ', r2)
```

```
Root mean squared error:  6.552468174207496
R2 score:  0.8604825502422389
```