Teaching Assistants Demand Prediction

Andrew Lau

BSc Computer Science

Submission Date: 26th April 2024

Supervisor: Sarah Sanders

*This report is submitted as part requirement for the BSc Degree in Computer Science at UCL. It is substantially the result of my own work except where explicitly indicated in the text.*

*The report may be freely copied and distributed provided the source is explicitly acknowledged.*

Abstract

The recruitment of Postgraduate Teaching Assistants (PGTAs) around University College London (UCL) Computer Science Department has been an ambiguous process with no clear metrics that can predict demand accurately. Despite the obvious answers of number of students, workload of the module and the difficulty of the syllabus, there is no linear relation between these metrics and the demand of PGTAs after plotting graphs to look at the results. Experimenting with AI models and different statistical tool has also proved to be inaccurate due to the unpredictability nature of modules. Hence, this project research deeper into what creates demand for PGTAs and how it can be better predicted for future references.

**Contents List**

1. Introduction
   1. Aims & Goals
   2. Deliverables
2. Background Information
3. Requirements and Analysis
4. Design and Implementation
5. Results Evaluation
6. Conclusion

**1 Introduction**

**Problem Statement**

At University College London (UCL) Computer Science Department, the recruitment of Postgraduate Teaching Assistants (PGTAs) is done manually through a recruitment process filled with ambiguity around the demand for PGTAs. For modules that are new, there isn’t a tried-and-tested method or official guideline on how to measure demand for PGTAs and it is usually done based on a review of the module content and intuition. This often leads to inaccurate estimation of workforce needed and inefficient allocation of resources within the department. The research is initially done around features that directly affect the demand of PGTAs such as number of students enrolled, coursework to exam ratio, module difficulty, etc. However, due to inconsistent datasets and unpredictability, further research is done in the field of AI prediction models and Natural Language Processing to analyse the module content and structure in more detail.

**1.1 Aims & Goals**

The project aims to better understand the demand PGTAs through analytical tools that drives data-driven decisions in the recruitment process. The analytical tool hopes to bring new insights into PGTA recruitment to realise its potential for ensuring that resources are better allocated to improve the learning experiences of students at UCL. Here are the aims and goals that should be achieved by the end of the project:

* Learn about the problems behind PGTA recruitment.
* Determine the factors that affects the demand of PGTA by researching into the recruitment process.
* Process data from module information and past records to create a combined dataset for training machine learning models.
* Accurately predict the necessary number of PGTAs to hire for a new and unfamiliar module.

**1.2 Deliverables**

The project submission would include the source files a documented and functional Dash app including:

1. Dashboards and analytical tools built using Plotly, including different types of graphs that provide a visual representation of the dataset:
   1. PGTAs Requested vs PGTAs Recruited Graph
      * Provides insights into which modules have inaccurate estimation of PGTA demand and its scale
   2. Students Enrolled vs PGTAs Recruited Graph
      * To determine whether the number of students enrolled affect the number of PGTAs recruited for the academic year 2022-2023 (only data for one year can be found)
   3. Exam-Coursework Ratio vs PGTAs Recruited Graph
      * To determine whether the weight of coursework and exams affect the number of PGTAs recruited
   4. Module Delivery Code vs PGTAs Recruited Graph
      * To determine whether module delivery code (modules taken by students in different years of study. In other words, difficulty of the module) affect the number of PGTAs recruited
2. Machine learning models drawn from Scikit-learn libraries.
   1. Several different machine learning models has been trained and tested with different variants. The one that was chosen as the final prediction model is a ridge regression model with cross-validation. A more detailed analysis can be found at section \_\_\_
3. Natural Language Processing models drawn from Scikit-learn and NLTK libraries

**2 Background Information**

**2.1 Background Research**

The project kicked-off with a brief of the problem statement given by my project supervisor, explaining the problem with ambiguity around the hiring of PGTAs.

**2.1.1 Technology Stack**

Selecting the technology stack for a project can be dependent on specific requirements of the project, such as the complexity of data, real-time processing needs, interactivity level, and the intended user’s technical proficiency. My supervisor will be the intender user after completion of this project and hence, the technology stack chosen must suit her technical proficiency and preferences. Hence, the main requirements of the technology stack include ease of use, Python based and freely available. A few different platforms that meet the main requirements was considered for the implementation of this project. Their strengths and weaknesses in different areas of the project are considered carefully to enable a successful implementation of this project. Here are the platforms considered and their evaluations:

**Grafana**

* Best for real-time monitoring and observability
* Specialises in complex time-series data visualisation
* Offers extensive customisation options through a wide range of plugins

**Streamlit**

* Ideal for quickly creating data apps, reports, and dashboards
* More suitable with static data with smaller datasets
* Easy to set up, configure and deploy with minimal infrastructure requirements
* Limited in terms of real-time capabilities and complex interactive features

**Plotly (Dash)**

* Focuses more on data visualisation
* Has better graphing tools
* Better suited for advanced interactivity

**Jupyter Notebooks**

* Interactive computing environment that supports live code, equations, visualizations, and narrative text.
* Highly popular in the data science community for exploratory data analysis and prototyping.
* Integrates well with Python libraries like Matplotlib, Seaborn, Plotly for visualization.

In conclusion, Streamlit and Dash are the two libraries that is most suitable for this project as they both offer data analysis and visualization capabilities. Dash is chosen as it is more flexible in nature compared to Streamlit, allowing more customisations within the app.

**2.1.2 Data Sources**

* Write about how data are given, and where it has been collected, the ethics behind retaining data and accessing them.
* The data are sourced by my supervisor in the format of excel sheets. The data are collected and recorded manually when students signed up to be a teaching assistant.
* The data provided did not contain any sensitive/personal information and hence, the ethical report is not needed to address the ethical conducts within this project

­­­

The main data source of this project is my supervisor. She is involved in the Postgraduate Teaching Assistant (PGTA) recruitment and collects data on module information cruicial for the implementation of this project. The data sheets provided are listed as below and five rows of data in each dataset is displayed for reference:

**Cap and Actual Students Dataset**

* This dataset contains data on the capacity of each module and the number of students who took each module

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module Code | Module Title | 2023-24 cap | 2022-23 actual students | Notes |
| COMP0002 | Principles of Programming | 170 | 170 | NaN |
| COMP0003 | Theory of Computation | 170 | 171 | NaN |
| COMP0004 | Object-Oriented Programming | 170 | 171 | NaN |
| COMP0005 | Algorithms | 170 | 171 | NaN |
| COMP0007 | Directed Reading | 27 | 18 | NaN |

**Module Assessment Dataset**

* This dataset contains data of each modules on their examinable components such as courseworks, projects, examinations, assessments, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| Module Code | Module Title | Assessment Type Name | Assessment Weight |
| COMP0002 | A4U | Exam (In Person Written) (Centrally Managed) | 90 |
| COMP0002 | A4U | Coursework | 5 |
| COMP0002 | A4U | Coursework | 5 |
| COMP0010 | A5U | Group project | 50 |
| COMP0010 | A5U | Coursework | 50 |

*\* Due to width limitation, columns are simplified and not all columns are included to allow readability \**

**New Module Dataset**

* This dataset contains the list of modules indicating whether each module is a new module in the year 2023. This dataset is not currently being used for analysis or decision-making of any kind but it might be looked upon further.

|  |  |  |
| --- | --- | --- |
| Module Code | Module Title | Is module new in 2023 |
| COMP0008 | Computer Architecture and Concurrency | FALSE |
| COMP0009 | Logic | FALSE |
| COMP0010 | Software Engineering | FALSE |
| COMP0011 | Introductory Mathematics for Computer Science | FALSE |
| COMP0012 | Compilers | TRUE |

*\* Due to width limitation, columns are simplified and not all columns are included to allow readability \**

**PGTA Requested and Recruited Dataset**

* This dataset contains the number of PGTAs requested and recruited for each module from the year 2021 to 2024.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Module Code | 2023-24 requested | 2023-24 recruited | 2022-23 requested | 2022-23 recruited | 2021-22 requested | 2021-22 recruited |
| COMP0002 | 150 | 100 | 150 | 115.5 | 150 | 67 |
| COMP0003 | 140 | 64 | 140 | 124 | 140 | 140 |
| COMP0004 | 360 | 94 | 270 | 166 | 270 | 259 |
| COMP0005 | 180 | 120 | 120 | 134 | 120 | 80 |
| COMP0007 | 70 | 0 | 70 | 70 | 70 | 45 |

*\* Due to width limitation, columns are simplified and not all columns are included to allow readability \**

**Job Description Dataset**

* This dataset contains the job description of modules and the PGTA hours needed

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Select module | PGTA hours excl. marking | Marking hours excl. exam | Marking hours for exam | Duties |
| COMP0027 | 120 | 80.0 | 0.0 | Supporting scheduled sessions (computing lab / tutorial / class etc ), Providing student support (e.g. Moodle Q&A, office hours), Preparing lab/tutorial/class activities, Marking - other (e.g. coursework, coding activities, in class tests, formative assessment, etc) |
| COMP0113 | 100 | 60.0 | 0.0 | Supporting scheduled sessions (computing lab / tutorial / class etc ), Providing student support (e.g. Moodle Q&A, office hours), Facilitating student teams (e.g. projects), Preparing lab/tutorial/class activities, Marking - other (e.g. coursework, coding activities, in class tests, formative assessment, etc) |
| COMP0005 | 120 | 60.0 | 0.0 | Supporting scheduled sessions (computing lab / tutorial / class etc ), Marking - other (e.g. coursework, coding activities, in class tests, formative assessment, etc) |
| COMP0147 | 194 | 0.0 | 50.0 | Supporting scheduled sessions (computing lab / tutorial / class etc ), Providing student support (e.g. Moodle Q&A, office hours), Preparing lab/tutorial/class activities, Marking - end of year exam (term 3) |
| COMP0009 | 95 | 0.0 | 0.0 | Supporting scheduled sessions (computing lab / tutorial / class etc ), Providing student support (e.g. Moodle Q&A, office hours), Marking - other (e.g. coursework, coding activities, in class tests, formative assessment, etc) |

*\* Due to width limitation, columns are simplified and not all columns are included to allow readability \**

1. **Data Handling and Database Implementation**

**3.1 Data Processing**

The provided dataset required extensive preprocessing due to the presence of missing values and data in inconsistent formats. The dataProcessing.py file includes a suite of functions crucial for data preparation, enabling analysis and modeling. Each function plays a role in transforming raw data into a structured format conducive to analytics and predictive modeling, ensuring the usability of the dataset. Here's a detailed look at these functions:

1. **no\_data\_modules (df, col1, col2)**

* This function identifies modules with 'No data found' entries in specified columns. It returns a list of module codes where either of the two specified columns contains 'No data found'. It is used for listing modules with incomplete data in the PGTAs recruited and requested columns for a selected year.

1. **create\_combined\_variables\_df (df\_moduleAssessmentData, df\_capVsActualStudents, df\_requestedVsRecruited)**

* Combines data from multiple DataFrames (df\_moduleAssessmentData, df\_capVsActualStudents, df\_requestedVsRecruited) to create a single DataFrame. This new DataFrame includes Module Code, Number of Students, PGTAs Recruited, Exam:Coursework Ratio, and Delivery Code. It filters modules based on their presence in all source DataFrames and sorts them by Exam:Coursework Ratio. It is used in plotting the graphs of each individual variables (Module Code, Number of Students, Exam:Coursework Ratio, and Delivery Code) against PGTAs recruited to gain insights on how each variables affect the number of PGTAs recruited.

1. **create\_coursework\_exam\_ratio\_column (df)**

* This function adds a new column 'Exam:Coursework Ratio' to the DataFrame by merging and calculating the weights of exam and coursework from different columns. It then drops duplicates and handles missing values. This column is essential to create the ‘Exam:Coursework Ratio vs PGTAs Recruited Graph’.

1. **split\_coursework\_exam\_ratio\_column (df)**

* Splits the 'Exam:Coursework Ratio' column into two separate columns ('Exam Weight' and 'Coursework Weight') and removes the original ratio column. This is processed so that each of the coursework weights and exam weights are used as features for training the machine learning model.

1. **handle\_missing\_data (df, columns)**

* Substitutes 'No data found' entries with 0 in specified columns and converts them to numeric format. This ensures data uniformity and prevents computational errors during analysis.

1. **handle\_nan\_data (df)**

* Fills NaN (not a number) entries with zero, addressing the issue of missing data. This also ensures data uniformity and prevents computational errors during analysis.

1. **column\_sum (df, column)**

* Calculates and returns the sum of values in a specified column of the DataFrame, providing aggregated data of total students, PGTAs hired and requested for each year.

1. **column\_average (df, column)**

* Calculates and returns the average of values in a specified column of the DataFrame, providing the average PGTA hours for each duty, allowing direct comparison.

1. **difference\_calculation (df, selected\_year)**

* Computes the difference between requested and recruited PGTAs for a given year and adds this as a new column in the DataFrame. This feature is pivotal in evaluating recruitment effectiveness and identifying gaps in estimation of PGTA demand for each year.

1. **set\_color (df)**

* Assigns a color based on the difference between requested and recruited PGTAs, with red colour indicating the demand was higher than expected while green colour indicates demand lower than expected, allowing easy data interpretation.

1. **load\_data (df)**

* Prepares features and target variables for modeling. It one-hot encodes the 'Delivery Code' column, merges this with the rest of the DataFrame, and returns the feature matrix (X) and target vector (y). This is fundamental for feeding data into predictive models.

1. **get\_total\_pgta\_hours (df)**

* Calculates the total PGTA hours required for each job description by summing the PGTA hours excluding marking, marking hours excluding end of year exam, and marking hours for the end of year exam. This function modifies the DataFrame with an additional column (PGTA hours) representing the total hours required, including all types of marking hours.

1. **split\_duties (duty)**

* Splits a string containing multiple duties separated by commas. It uses a stack to keep track of parentheses to avoid incorrect splits for commas within parentheses and returns a list of individual duties.

1. **get\_set\_of\_duties (job\_desc)**

* Extracts a unique set of duties from the job descriptions, serving as the basis for feature engineering.

1. **create\_feature\_vector (df, unique\_duties)**

* Transforms the 'Duties' column into a binary feature matrix. Each duty is given its own column, where a 1 indicates the duty's presence in a job description, and a 0 indicates its absence.

1. **filter\_base\_duty\_in\_duties (df, duty)**

* Filters a DataFrame based on the presence of a specified duty within the 'Duties' column of the DataFrame. It performs a case-insensitive search for the specified duty within each entry of the 'Duties' column, returning a new DataFrame that includes only those rows where the duty is found. The use of ‘*re.escape(duty)’* escapes special characters so that only the actual text of the duty is being matched instead of a regex pattern.

1. **download\_nltk\_resources ()**

* Downloads the necessary NLTK (Natural Language Toolkit) resources, which include tokenizers, stop words, wordnet, and a POS tagger. It is called before using any NLTK functionality to ensure that the required resources are available.

1. **tokenize\_text (text)**

* Tokenizes the input text into a list of words or tokens.

1. **remove\_stopwords (tokens)**

* Removes common words that do not carry significant meaning from a list of tokens.

1. **get\_wordnet\_pos (word)**

* Maps POS (Part of Speech) tags to the format accepted by the WordNetLemmatizer library for lemmatization.

1. **lemmatize\_tokens (tokens)**

* Lemmatizes a list of tokens, reducing them to their base or dictionary form.

1. **Preprocess\_text (text)**

* Applies all the preprocessing steps (tokenization, stopword removal, lemmatization) to the input text.

1. **preprocess\_text\_list (text\_list)**

* Calls the *preprocess\_text* function on a Pandas Series containing text descriptions to be preprocessed.

1. **vectorize\_documents (preprocessed\_text)**

* Vectorizes a series of preprocessed documents using TF-IDF (Term Frequency – Inverse Document Frequency) vectorization.

**3.2 Database (SQLite)**

The implementation of this database is crucial for our predictive models, enabling the analysis of relationships between module characteristics and TA allocation needs. By leveraging a well-defined schema, we ensure that our data is robust and reliable.

**3.2.1 Database Comparison**

SQLite was chosen as the database management system for this project due to its serverless architecture, which provides ease of use and convenience for smaller-scale applications. As a self-contained, file-based database, SQLite provides a lightweight solution without the need for a separate server process, minimising setup and administrative tasks. This also results in low-latency access to the data, as read/write operations do not involve network communication or complex protocols associated with client-server Database Management Systems (DBMS).

SQLite has limitations in handling concurrent write operations, user management or large-scale data processing needs. However, it shouldn’t be a concern as the requirements of this project is mainly a database that supports simple CRUD operations and this web application is intended to be used by one user only. Hence, SQLite is an excellent choice.

Traditional SQL databases like PostgreSQL and MySQL are designed for a different set of requirements. They excel in environments where large-scale data handling, complex transactions, and high concurrency are common. They offer extensive features for user management, data security, and advanced query optimizations. However, these systems require more resources in terms of hardware and maintenance. They may introduce more latency due to the client-server model, which can impact the speed of operations.

**3.2.2 Database Implementation**

The SQLite relational database is implemented composing of five tables, each capturing different aspects of module information. The PRIMARY KEY in each table ensures data integrity and provides a unique identifier for each record. These tables and their properties are defined as follows:

|  |  |  |
| --- | --- | --- |
| Job Descriptions | Number of Students | requested\_vs\_recruited |
| PRIMARY KEY (id)  module\_code VARCHAR  module\_title VARCHAR  number\_of\_TA INTEGER  duties VARCHAR  total\_hours INTEGER | PRIMARY KEY (id)  module\_code VARCHAR  module\_title VARCHAR  cap\_23\_24 INTEGER  actual\_22\_23 INTEGER  notes TEXT | PRIMARY KEY (id)  module\_code VARCHAR  module\_title VARCHAR  requested\_23\_24 INTEGER  recruited\_23\_24 INTEGER  requested\_22\_23 INTEGER  recruited\_22\_23 INTEGER  requested\_21\_22 INTEGER  recruited\_21\_22 INTEGER |

|  |  |  |
| --- | --- | --- |
| module\_assessment | average\_pgta\_hours | combined\_variables |
| PRIMARY KEY (id)  module\_code VARCHAR  delivery\_code VARCHAR  module\_delivery\_period\_code VARCHAR  exam\_weight INTEGER  coursework\_weight INTEGER  exam\_coursework\_ratio VARCHAR | PRIMARY KEY (id)  duties VARCHAR  average\_hours | PRIMARY KEY (id)  module\_code VARCHAR  module\_title VARCHAR  number\_of\_students INTEGER  pgtas\_recruited INTEGER  exam\_coursework\_ratio VARCHAR  exam\_weight INTEGER  coursework\_weight INTEGER |

**3.2.2.1 SQLAlchemy**

SQLAlchemy is a Python SQL toolkit and Object-Relational Mapping (ORM) library, responsible for managing interactions with the SQLite database. It allows developers to work with Python objects rather than SQL queries, enhancing code readability and maintainability.

By integrating SQLAlchemy with SQLite, the project benefits from the ease of database handling and manipulation using SQLAlchemy's ORM feature, coupled with the efficiency and simplicity of a serverless database system, making it a great setup for Python-based applications with moderate data management requirements.

**3.2.2.2 Database Initialisation**

This section discusses a brief outline of the database workflow for initialisation.

1. **Model Declaration:** SQLAlchemy's ORM feature is used to initiate models such as JobDescription, RequestedVsRecruited, ModuleAssessment, etc. which define the structure of the tables within the SQLite database.
2. **Engine Creation:** A database engine is initiated through SQLAlchemy's *create\_engine* method, which establishes the connection to the SQLite database file. The *init\_db* function initialises the database where Base.metadata.create\_all(engine) creates all the defined tables.
3. **Session Management:** The session, an instance of the Session class created by the sessionmaker bound to the engine, manages interactions with the database by acting as a staging zone for all objects loaded into the database session.
4. **Data Manipulation:** CSV data is directly loaded into the database tables in the *load\_csv\_to\_database* function. This function iterates over a pandas DataFrame, creating instances of the corresponding model populated with data from each row, and adds these instances to the SQLAlchemy session.
5. **Session Commitment:** The *session.commit()* method is called to persist all staged changes to the database, translating the object state operations into corresponding SQL statements.
6. **Session Closure:** Upon committing the changes, the session is terminated via *session.close()* to free resources and maintain database integrity.
7. **Graph Analysis**

This section discusses the dynamics between the demand for Postgraduate Teaching Assistants (PGTAs) through a series of carefully plotted visualizations. The core components being analysed are the discrepancies between predicted and actual TA needs, the impact of student enrollment numbers, assessment structures, the correlation between module delivery codes, specific duties, and allocated PGTA hours. Each graph not only offers a standalone insight into specific facets of TA needs but also collectively provides a comprehensive overview. This analysis aims to explore the underlying patterns and anomalies within TA recruitment processes, offering a data-driven foundation for enhancing future predictive models.

1. **PGTAs Requested vs PGTAs Recruited Graph**

* This graph aims to provide insights into which modules have inaccurate estimation of PGTA demand and its scale.

A screenshot of a graph

Description automatically generated

.

.

.

A screenshot of a graph

Description automatically generated

**Analysis:** The bar graph illustrates the comparison between the number of PGTAs requested by modules and the actual number recruited. This visualization highlights modules with significant discrepancies, which suggests a potential misalignment between anticipated and actual TA needs. Modules with a higher number of recruited PGTAs than requested may indicate either an underestimation of TA requirements or an adjustment to increased student demand or course complexity.

1. **Module History Graph**

* Building upon the previous graph, this visualization presents an alternative view of the history of PGTAs requested vs PGTAs recruited for a selected module from the dropdown menu, allowing direct comparison between different modules. It also shows the total number of PGTAs recruited and requested across all modules for the past three years.

A screenshot of a graph

Description automatically generated

1. **Students Enrolled vs PGTAs Recruited Graph**

* This graph aims to determine whether the number of students enrolled affect the number of PGTAs recruited for the academic year 2022-2023 (only data for one year can be found)

A graph with blue dots

Description automatically generated

**Analysis:** The scatter plot suggests no clear linear relationship between the number of students enrolled in a module and the number of PGTAs recruited, indicating that student numbers are not the sole determinant of TA recruitment. Several outliers suggest that other factors such as module complexity or administrative decisions may play a role.

1. **Exam-Coursework Ratio vs PGTAs Recruited Graph**

* This graph aims to determine whether the weight of coursework and exams affect the number of PGTAs recruited.

A graph with numbers and text

Description automatically generated

**Analysis:** This graph shows a distribution of PGTAs recruited across different exam-coursework ratios. A higher concentration of points towards extreme ratios (e.g., 100:0 or 0:100) may suggest that modules with a singular assessment focus either on exams or coursework might have more predictable TA needs. However, the spread across the spectrum indicates that the ratio alone does not dictate TA recruitment patterns.

1. **Module Delivery Code vs PGTAs Recruited Graph**

* This graph aims to determine whether module delivery code (modules taken by students in different years of study. In other words, difficulty of the module) affect the number of PGTAs recruited.

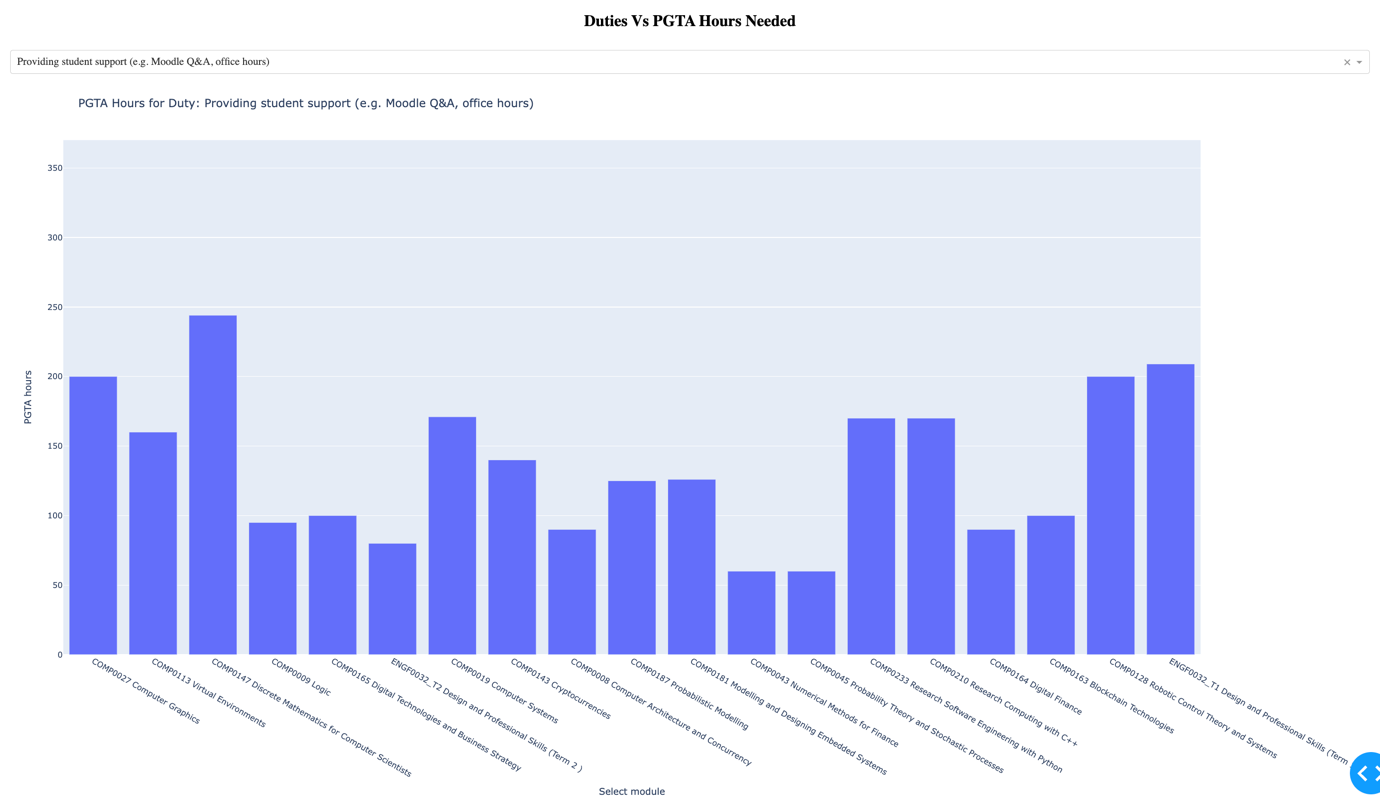
A white box with blue dots

Description automatically generated

**Analysis:** The final graph examines the relationship between module delivery codes and the number of PGTAs recruited. Variability across delivery codes could suggest that the nature of the module, potentially its complexity or the year of study it's catered to, might influence TA recruitment. However, the data presents a diverse pattern, suggesting a more intricate interplay of factors beyond just the delivery code.

1. **Duties vs PGTA Hours Graph**

* This graph aims to determine the relationship between specific duties assigned within modules and the corresponding PGTA hours allocated.



**Analysis:** The 'Duties' column encompasses a mix of six base duties:

*1. Providing student support (e.g. Moodle Q&A, office hours)*

*2. Facilitating student teams (e.g. projects)*

*3. Marking - other (e.g. coursework, coding activities, in class tests, formative assessment, etc)*

*4. Marking - end of year exam (term 3)*

*5. Preparing lab/tutorial/class activities*

*6. Supporting scheduled sessions (computing lab / tutorial / class etc )*

For each selected duty from the dropdown menu, a bar graph shows the PGTA hours for all modules containing the selected duty in its job description. This visualization helps identify which duties demand more or less PGTA time, reflecting on the potential intensity or complexity of these duties. However, the variation in PGTA hours for modules sharing common duties indicates that direct comparison of duty intensity is not straightforward. To address this, an aggregated view is plotted in the subsequent graph.

1. **Duties vs Average PGTA Hours Graph**

* Building upon the previous graph, this visualization presents the average PGTA hours for modules that contain each base duty, streamlining direct comparisons across duties.

A graph of blue rectangular shapes

Description automatically generated with medium confidence

**Analysis:** The graph reveals that the duty 'Supporting scheduled sessions (computing lab / tutorial / class etc.)' requires the most PGTA hours on average, while 'Facilitating student teams (e.g., projects)' demands the least. This insight is pivotal for refining our prediction model; understanding the differential demands of various duties allows for more accurate forecasts of PGTA hour requirements.

**Results and Evaluation**

* Across all visualizations, it is evident that predicting TA demand is multifaceted and complicated, with no single variable providing a complete understanding.
* The disparities in requested versus recruited PGTAs highlight the need for refined predictive models that can accommodate a range of influencing factors.
* The lack of a strong correlation in some graphs suggests that further qualitative data, such as module content analysis or instructor feedback, might be necessary to fully understand TA recruitment needs.
* These analyses underscore the complexity of academic resource planning and the potential for data-driven approaches to enhance decision-making processes.

**Conclusion**

The visualizations provide valuable insights into the factors influencing TA recruitment. However, they also highlight the complexities in academic administration and the potential benefits of adopting comprehensive analytic strategies to improve accuracy in PGTA demand.

**3.3 Machine Learning Models**

This section discusses the machine learning models used within this project to conduct analysis with documented results and evaluation.

**3.3.1 Linear Regression and Ridge Regression**

**Linear Regression** is a statistical method that models the relationship between a scalar dependent variable y and one or more independent variables (or 'predictors') X. The core idea is to obtain a linear equation that predicts y as accurately as possible from X. It is based on the ordinary least squares (OLS) estimator and assumes a linear relationship between the variables.

**Ridge Regression** is an extension of linear regression where the loss function is modified to minimize the complexity of the model. This is achieved by adding a penalty term to the OLS loss function, which is proportional to the square of the magnitude of the coefficients. Ridge Regression aims to prevent overfitting by penalizing large coefficients.

The linear\_regression and ridge\_regression functions defined in the script are designed to instantiate the respective regression models from the scikit-learn library.

* **linear\_regression(X, y):** This function configures a RandomForestRegressor from the scikit-learn library with a fixed number of estimators and a set random state for reproducibility.
* **ridge\_regression(X, y):** It sets up a Ridge regression model with a regularization strength alpha and a specified random state.
* **train\_model(X, y, model\_type):** Depending on the model type ('ridge' or 'linear'), this function trains the respective regression model on the dataset.
* **save\_model(model, filename):** This utility saves the trained model to the disk, allowing for later retrieval and inference.

Each function accepts the features X and the target variable y and returns a model object configured with a predetermined number of estimators (n\_estimators=100) and a random state (random\_state=42) to ensure reproducibility. The script proceeds to load and preprocess the dataset using various functions from the data\_processing module. After preparing the data, it trains the specified model, performs K-Fold cross-validation to estimate the model's performance, and saves the model. The save\_model function saves the trained model as a .pkl file to the specified filename, ensuring the model's persistence for future predictions. The function train\_model determines which regression model to train based on the model\_type parameter, which can be either 'ridge' or 'linear'.

**Results and Evaluation**

The performance of the two models was evaluated using a 5-fold cross-validation approach, with RMSE as the performance metric. The summarized results are as follows:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model Type | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 | Mean RMSE | Standard Deviation |
| Ridge | 116.26 | 83.91 | 94.65 | 89.18 | 96.46 | 96.09 | 11.00 |
| Linear | 123.72 | 88.99 | 100.68 | 103.74 | 91.19 | 101.66 | 12.34 |

**Conclusion**

The Ridge Regression model outperformed the Linear Regression model with a lower mean RMSE, indicating better average performance across all folds. The standard deviation of the RMSE scores, which reflects the variability in model performance across different data splits, was also lower for the Ridge model, suggesting more consistent performance.

**3.3.2 Prediction Prompt**

The prediction prompt represents a critical component of the research project, bridging the gap between user input and sophisticated machine learning models. This section documents the implementation of a user-friendly interface that allows for the input of relevant features to predict the number of Postgraduate Teaching Assistants (PGTAs) required.

**Model Selection**

* The selected model from the model\_type variable is loaded from the models directory using the load\_model function.

**User Interface**

* A web-based interface is implemented using Dash, a Python framework for building analytical web applications.
* The interface includes input fields for 'Number of Students', 'Exam Weight', 'Coursework Weight', and 'Delivery Code'.
* A 'Predict' button initiates the prediction process.

**Data Preparation**

* User inputs are captured and formatted into a DataFrame that matches the model's expected input structure.
* The 'Delivery Code' is transformed into dummy variables to align with the model’s training data.
* Missing columns in the input data (not present during model training) are filled with zeros to maintain consistency.

**Prediction**

* The formatted input data is passed to the machine learning model for prediction.
* The model outputs the predicted number of PGTAs required, which is displayed to the user.

**Error Handling and Validation**

* The system currently does not include error handling on incorrect or incomplete input but it will be looked upon in future developments
  1. **Generalised Additive Model**

This section dives into another method of PGTA recruitment predictions by deploying a Generalized Additive Model (GAM). GAMs offer a more flexible approach to modeling complex, non-linear relationships in data compared to linear regression, which is more straightforward but less adaptable to non-linearity. The focus is on refining the dataset for predictive accuracy and assessing the model's performance using root mean square error (RMSE) as a metric. A GAM was constructed to predict PGTA requirements, considering factors such as student numbers and module delivery codes. The dataset was partitioned into training and testing subsets, maintaining a test size of 20%. The GAM is then fitted on the training data, leveraging spline functions to capture non-linear relationships.

**Results and Evaluation**

The analysis revealed a RMSE score of 131.68, higher than the mean RMSE of both linear and ridge regression models. This result indicates high inaccuracy of the model despite being trained on a non-linear dataset. In order to assess these results further, the next section compares GAMs and Linear Regression models.

**Generalized Additive Model (GAM) and Linear Regression Comparison**

The comparison between Generalized Additive Models (GAMs) and Linear Regression models is crucial in understanding the flexibility and applicability of these statistical techniques in various data analysis scenarios. Here's a detailed examination of their differences, along with evaluations:

|  |  |  |
| --- | --- | --- |
|  | GAM | Linear Regression |
| Flexibilty and Complexity | Offers the flexibility to model complex, non-linear relationships between predictors and the response variable.GAM manages complexity via smoothing functions and regularization | A structurally simpler model that assumes a linear relationship between these elements​​, which can be a limiting factor for accuracy if the true relationship is non-linear. |
| Regularization | Can inherently include regularization techniques to control model complexity | Requires external regularization methods like Ridge or Lasso |
| Sample Size Requirements | Often require larger sample sizes to capture non-linear trends effectively | Less demanding due to its simplicity​​ |
| Visualizations | Visualization of smooth functions in GAMs can be particularly helpful in understanding the relationships within the data | Difficult to be represented in graphs or other visual representations |

**Conclusion**

This research highlights the significance of conplex data processing and the effectiveness of GAMs in forecasting non-linear datasets. However, it is difficult to capture non-linear relationships and since the dataset it was trained on has only limited data, the desired accuracy cannot be achieved.

**3.5 Natural Language Processing**

**Introduction**

Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and humans through natural language. It uses techniques that extract key features and entities from textual data to enable predictive capabilities…

The linear regression models were built on data such as number of students, coursework:exam weightage ratio, etc. After careful evaluatiuon, they are not the most accurate representation of PGTA demand. Hence, a NLP model is built to determine whether the job decriptions of PGTAs, which include their duties, affect their demand. This model aims to predict the number of hours needed based on their duties.

The implementation of NLP models can be a complex process and a successful implementation requires careful consideration of the data, model selection and techniques. In the context of this part of the project, linguistic analysis is needed before feeding data into a machine learning model, hence NLTK is used for the preprocessing steps as it excels in providing tools for detailed text processing and scikit-learn for model building and evaluation.

**3.5.1 Prediction Using TF-IDF**

**Data Preprocessing**

This step involves applying text preprocessing to the "Duties" column, which includes tokenization, stopwords removal, stemming/lemmatization, and vectorization. This process converts the text into a numerical format that can be used by machine learning models.

**Tokenization**

* Tokenization is the process of splitting text into units called tokens, which can be words, sentences, or subwords. This is the first step in text preprocessing and is crucial for understanding the structure of the text. This is done using nltk.tokenize. word\_tokenize.

**Stopwords removal**

* Stopwords are common words like "and", "the", "is", etc., that are usually removed because they appear frequently in the text but don't carry significant meaning for analysis or modeling. This is done using nltk.corpus.stopwords to get a list of stopwords for English.

**Stemming and lemmatization**

Both processes aim to reduce words to their base or root form, but they do so differently:

* Stemming chops off word prefixes and suffixes indiscriminately, which might result in non-existent word forms but reduces the complexity of the textual data.
* Lemmatization considers the morphological analysis of the words, aiming to remove inflectional endings only and return the base or dictionary form of a word, known as the lemma.
* This is done using nltk.stem.PorterStemmer and nltk.stem.WordNetLemmatizer

**Vectorization**

Vectorization is the process of converting text into numerical data (vectors) so that machine learning algorithms can understand it. There are several methods to vectorize text:

* **Term Frequency-Inverse Document Frequency (TF-IDF):** Reflects how important a word is to a document in a collection or corpus. It not only counts the frequency of words but also scales down the impact of frequently occurring words across documents.
* **Word Embeddings**: Represents words in a high-dimensional space where the position of each word is learned from text based on its surrounding words. Pre-trained models like Word2Vec, GloVe, or embeddings from language models like BERT can be used.

After careful consideration, the TF-IDF method is used by importing the The sklearn.feature\_extraction.text.TfidfVectorizer library because it fits the dataset. Since we are only capturing the entities within the dataset, the ordering of text does not matter and hence, it is irrelevant to the count vectorization and word embedding methods. Here’s a more detailed explaination of TF-IDF:

TF-IDF is a statistical measure used to evaluate the importance of a word to a document in a collection or corpus. It is often used in text mining and information retrieval to weigh the frequency of words by how common they are across documents, thus helping to adjust for the fact that some words appear more frequently in general.

* **Term Frequency (TF):** This measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:
* **Inverse Document Frequency (IDF):** This measures how important a term is. While computing TF, all terms are considered equally important. However, certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scaling up the rare ones, by computing the following:

with the logarithm scale used to dampen the effect of IDF. If a term appears in all documents, its IDF value becomes 0 (since log(1) = 0), showing that the term is not unique or important.

The TF-IDF value is simply the multiplication of TF and IDF:

This value increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words are generally more common than others.

**Model Selection and Training**

* A model is chosen based on the nature of the data. The aim is to predict a numerical value (hours needed) and given that the relationship between text features and hours is linear, the regression models are the most suitable candidates.
* Training is done by first dividing the data into training and test sets with cross-calidation to ensure the model’s performance is robust across different subsets of the data. The model is trained using the vectorized "Duties" as input features and "PGTA hours excluding marking" as the target variable.

**Model Comparison**

A comparative analysis of three regression models is conducted to predict the required PGTA hours based on job descriptions. The dataset comprised 35 entries, each with an associated 'Duties' text column and a numerical 'PGTA hours excluding marking' target variable. Our objective was to determine which model would provide the most accurate predictions as measured by the Root Mean Squared Error (RMSE) across a 3-fold cross-validation process.

**Models and Preprocessing:**

Each model was paired with a Term Frequency-Inverse Document Frequency (TF-IDF) vectorization to preprocess the text data.

The cross-validation average RMSE and standard deviation for each model were as follows:

|  |  |  |
| --- | --- | --- |
| Model | Average RMSE | Standard Deviation |
| Ridge Regression | 52 | 13 |
| Linear Regression | 66 | 0.6 |
| Random Forest Regressor | 61 | 8 |

**Evaluation**

Ridge Regression exhibited the lowest average RMSE of 52, indicating the best average performance across the folds. However, it also had the highest standard deviation (13), which suggests a variability in performance across different subsets of the data. This could be indicative of model sensitivity to the specific data it is trained on, potentially leading to overfitting.

Linear Regression had the highest average RMSE (66), implying less accurate predictions on average. Nevertheless, it demonstrated the lowest standard deviation (0.6), suggesting consistent performance across different data splits. The consistency could indicate a more stable model that generalizes better, but it may also mean that the model is not fitting the data as closely as the Ridge Regression model.

Random Forest Regressor presented a average RMSE similar to Linear Regression (61) but with a higher standard deviation (8). This indicates a moderate level of both accuracy and consistency. The Random Forest model might be striking a balance between fitting the data and generalizing across different data subsets.

Given the small dataset size (31 rows), the high standard deviation observed with Ridge Regression might be of concern, despite its lower average RMSE. The Linear Regression model, while less accurate, may be more reliable when applied to unseen data due to its consistency. The Random Forest Regressor appears to offer a middle ground between the two and hence, yields the best overall result.

**3.5.1 Prediction Using Feature Engineering**

**Introduction**

Feature Engineering is a pivotal aspect of the predictive modeling process, especially when dealing with natural language data such as job descriptions. Binary encoding is a form of feature engineering where textual data is converted into a numerical format that machine learning models can interpret and learn from. Binary encoding is particularly suited for this context because the job descriptions may involve a wide variety of base duties, but each description mentions only a subset. Binary encoding efficiently represents this sparse data. Additionally, it allows for straightforward comparisons across job descriptions, enabling analysis of which duties commonly lead to higher or lower PGTA hours.

**Data Preprocessing**

This step involves applying text preprocessing to the "Duties" column, which includes getting the total PGTA hours including marking, identifying the set of base duties and creating a feature vector. These functions collectively prepare the dataset for predictive modelling:

* **get\_total\_pgta\_hours** aggregates various types of PGTA hours into a single total, simplifying the target variable for prediction.
* **get\_set\_of\_duties** identifies unique duties within job descriptions, essential for creating a comprehensive list of potential features.
* **create\_feature\_vector** transforms the list of duties into a binary vector, indicating the presence (1) or absence (0) of each duty in job descriptions.

**Model Training**

With the features predefined, a RandomForestRegressor model is trained on the dataset. This model is chosen for its ability to handle non-linear relationships and interactions between features. From previous experimentations, this regressor model yields the best overall performance when compared to linear regression and ridge regression. Cross-validation with KFold is utilized to assess the model's performance, ensuring its robustness across various data subsets.

The mean RMSE and standard deviation across the folds provide insights into the model's prediction accuracy and consistency.

**Evaluation**

The cross-validation average Root Mean Square Error and Standard Deviation for the model is:

|  |  |  |
| --- | --- | --- |
| Model | Average RMSE | Standard Deviation |
| Random Forest Regressor | 61 | 6 |

Random Forest Regressor presented a average RMSE of 61 with a standard deviation of 6, indicating a moderate level of both accuracy and consistency.

**Conclusion**

Feature engineering through binary encoding stands out as a transformative step in the prediction pipeline. By converting the nuanced textual data of job descriptions into a clear, structured format, we enable the machine learning model to make informed predictions about PGTA hours. This approach underscores the importance of a nuanced understanding of the data's characteristics in the context of predictive modeling, thereby enhancing the quality and accuracy of the predictions made for PGTA hour allocation.

**3.6 Prediction Prompts**

This section documents the implementation of a user-friendly interface that allows for user input to get an estimate of the PGTA hours required by feeding the input into machine learning models.

**3.6.1 Ridge Regression Prompt**

This interface utilizes a trained Ridge Regression model, which is a variant of linear regression with regularization to prevent overfitting. The prompt is part of a web application built using the Dash framework, enabling interactive and user-friendly prediction based on input variables related to a module. The prompt provides fields for users to input relevant predictors such as the number of students enrolled in a module, the weight of exams, the weight of coursework, and the delivery code associated with the module.

A diagram of a graph

Description automatically generated

**A screenshot of a computer

Description automatically generated**

When the predict button within the Dash interface is clicked (indicated by n\_clicks > 0), the *linearRegressionPredictor* function proceeds to create a DataFrame with the provided inputs. The Delivery Code input is one-hot encoded to match the format used during model training. The function ensures that all feature columns expected by the model are present in the input DataFrame, adding any missing columns with a default value of 0. The columns are also reordered to match the training data's order. The Ridge Regression model loaded from file is then used to predict the number of PGTAs based on the input data.

**3.6.2 TF-IDF Vectorisation Prompt**

TF-IDF vectorization can extract semantic features from job descriptions, which are effective in forecasting outcomes like the estimated hours required for a task. The *vectoriserPredictor* function is designed to handle user interactions with a web application built with the Dash framework. The function aims to predict the number of Postgraduate Teaching Assistants (PGTAs) hours required based on a set of selected duties using a pre-trained machine learning regression model.

A diagram of a diagram

Description automatically generated

A white background with black text

Description automatically generated

When the predict button within the Dash interface is clicked (indicated by n\_clicks > 0), the *vectoriserPredictor* function concatenates the user-selected duties into a single string. This string undergoes preprocessing where it is tokenized, cleaned of stopwords, lemmatized, and then reconstructed into a processed string. This preprocessed text is then vectorized using a TF-IDF vectorizer, a critical step that translates the text into a numerical form, necessary for the regression model's input.

**3.6.2 Feature Engineering Prompt**

The feature engineering process begins with the collection of duties described in job descriptions for PGTAs. Each duty is encoded as a binary feature indicating its presence or absence in the description. A predictive linear model is then applied to this binary vector to forecast the required PGTA hours.

The *featureEngineeringPredictor* function is designed to handle user interactions with a web application built with the Dash framework. The function aims to predict the number of Postgraduate Teaching Assistants (PGTAs) hours required based on a set of selected duties using a pre-trained machine learning regression model.

A diagram of a diagram

Description automatically generated

A screenshot of a test

Description automatically generated

When the predict button within the Dash interface is pressed (indicated by n\_clicks > 0), the function initialises a dictionary with all possible duties set to 0. It then updates this dictionary to set the selected duties to 1, creating a binary representation of the duties. A pandas DataFrame (input\_df) is created from the dictionary, which serves as the input for the machine learning model. The function ensures that the input DataFrame contains all the features that the model was trained on. If any are missing, it adds these features to the DataFrame and sets them to 0, indicating the absence of those duties in the current prediction context. The columns in the input DataFrame are reordered to match the order expected by the trained model, which is necessary for the model to make an accurate prediction. The model's predict method is called with the prepared DataFrame, and the first prediction is accessed. This prediction indicates the estimated number of PGTAs required and is displayed in the Dash application.

**3.6.3 Unified Prompt**

Combining the ridge regression predictor with the feature engineering predictor and TF-IDF predictor into an integrated prediction system could potentially yield better results by leveraging both structured parameters (like number of students, exam weight, etc.) and unstructured data (job descriptions) in the predictions. This approach can provide a more comprehensive analysis of the factors influencing PGTA recruitment.

**4 Results Evaluation / Techniques Used**

**4.1 Cross-validation (CV)**

* A model usually creates a prediction function from the same set of data, which might lead to a bias towards that dataset. This might produce inaccuracies when prompted with un-seen data (a situation called overfitting). As the datasets provided in this project are small, they are more prone to overfitting and necessary measures are needed to prevent that.
* Cross-validation solves this problem by using the K-fold strategy. The dataset is split into k smaller datasets and the model is trained and tested k times on k-1 folds and 1-fold respectively. E.g. a CV with 5 folds running on a dataset of 100 data will produce 20 data per fold.
* Cross-validation is computationally done using the cross\_val\_score helper function on the dataset by splitting the data, fitting a model, and computing the score for k number of consecutive times (with different splits each time). The Root Mean Squared Error (RMSE) is used as a performance metric.
* The results of each of the selected number of folds are documented below:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| K - folds | Linear Regression | | Ridge Regression | | Feature Engineering | | TF-IDF Vectorizer | |
| Mean RSME | Standard Deviation | Mean RSME | Standard Deviation | Mean RSME | Standard Deviation | Mean RSME | Standard Deviation |
| 3 | 84.05 | 8.41 | 73.90 | 9.62 | 68.93 | 4.08 | 67.58 | 3.60 |
| 5 | 85.7 | 22.35 | 72.67 | 9.56 | 58.32 | 25.91 | 57.89 | 21.26 |
| 10 | 75.47 | 32.61 | 69.90 | 28.97 | 56.26 | 33.51 | 55.76 | 28.49 |
| 15 | 63.84 | 42.42 | 60.31 | 39.53 | 51.28 | 40.33 | 51.67 | 34.78 |

* Low Mean RMSE: This indicates that the model, on average, has a low prediction error, directly relates to the model's accuracy.
* Low Standard Deviation: This suggests that the model's performance is consistent across different subsets of the dataset, crucial for ensuring that the model is reliable and not just performing well on specific types of data.

**Conclusion**

Considering the trade-offs between accuracy and consistency, a model with a balanced mean RMSE and standard deviation is chosen for the context of predicting PGTA Hours. Hence, the optimal models are Linear Regression Model with 3-fold cross-validation, Ridge Regression Model with 5-fold cross-validation, Feature Engineering Model with 3-fold cross-validation and TF-IDF Vectorizer Model with 3-fold cross-validation.

**5 Conclusion**

**6 References**

1. Analytics Vidhya. (2023). Understanding Generalized Additive Models (GAMs): A Comprehensive Guide. Retrieved from <https://www.analyticsvidhya.com/blog/2023/09/understanding-generalized-additive-models-gams-a-comprehensive-guide/>. 24th January 2024

Things to be done

* literature review
* write more on the background
* combine the nlp model and the linear regression model
* integration testing

Issue / Challenges

* inaccurate representation of pgta hours when predicted using feature engineering
  + an input of 6 duties yield a lower pgta hour output compared to an input of 4 duties.
  + This is because the data that it is trained on mostly consists of 3 or 4 duties and wasn’t able to recognise 6 duties.