

Admissions Decision Machine Learning Integration Tool (ADMIT)

Katrina Stradford
Western Governors University
Student# 011334472

May 7, 2024

Table of Contents

Part A: Letter of Transmittal.....	2
Part B: Project Proposal Plan.....	4
Project Summary.....	4
Data Summary.....	4
Implementation.....	5
Timeline.....	6
Evaluation Plan.....	7
Resources and Costs.....	8
Part C: Application.....	10
Part D: Post-Implementation Report.....	11
Solution Summary.....	11
Data Summary.....	11
Machine Learning.....	15
Validation.....	19
Visualizations.....	21
User Guide.....	24
References.....	27

Part A: Letter of Transmittal

May 7, 2024

Harvey Daniels, Dean of Admissions
Alaska University
235 Gold St, Juneau, AK 99801

Dear Dean Daniels,

In response to your recent faculty address, I am writing to propose a solution to Alaska University's struggle with managing the increasing number of applicants we receive each year. During Alaska University's 85% increase in graduate applicants over the past three years, the admissions department has struggled to relay admissions decisions by the stated deadline. For the last two admissions cycles, this resulted in an increasing number of admitted students declining enrollment and attending other institutions. To maintain AU's ability to attract and secure top students for their educational programs, the admissions department must offload some of the manual processes within the current admissions process. As a Senior Machine Learning Engineer with experience developing tools to increase productivity in educational institutions, I believe I can assist AU in automating the admissions process.

Currently, the admissions staff is manually assigning admissions decisions to each applicant after scoring their statement of purpose letters. Not only is this time-consuming, but it also leaves space for bias to enter the admissions process. With the **Admissions Decision Machine Learning Integration Tool (ADMIT)**, Alaska University Admissions can automate the decision-making process. Using machine learning will reduce the time to review each application, get admissions decisions out to applicants quicker, and reduce admissions bias without heavily disrupting the current admissions workflow.

ADMIT is a web application that uses machine learning to review applicants based on their qualifications and automatically assign them admission or rejection to the university accordingly. Authorized users will use a user interface to bulk upload applications and render admissions decisions within minutes. They will also be able to render individual admissions decision and

review applicant metrics through dashboard charts. With a switch to ADMIT, the admissions team's only manual work would be to score the statement of purpose letters. ADMIT automates the rest of the admissions process and bases admissions decisions on the scores the university already prioritizes. These scores include the graduate readiness exam (GRE) score, statement of purpose (SOP) score, and cumulative grade point average (CGPA). The final implementation of the application will be executed within the university's existing faculty web portal. However, a standalone application will be available for demoing purposes. ADMIT will replicate past admissions decisions with a minimum of 75% accuracy, reduce time spent on the admissions process by at least 50%, and increase applicant satisfaction by 20%.

The data required to use the application can be sourced as a CSV file from the university's applicant tracking software. To ensure the university's continued FERPA and GDPR compliance, the application will be password-protected and all data passing through the application will be encrypted. To maximize the university's efforts to minimize bias in the admissions process, the application removes all personal identifying information for each applicant. Decisions are made solely based on each applicant's merit.

Implementation of the application is estimated to cost \$51,175 over a 9-week completion period. This implementation timeline includes requirements analysis, data analysis, development of the machine learning model and dashboard application, accuracy evaluation, and deployment. The projected cost includes the labor, hardware, software, and hosting needed to fully deploy the application. We propose a project start date of 07/29/2024, with deployment on 09/27/2024.

With my experience in EdTech and Machine Learning, I am confident I can manage the development and deployment of ADMIT within the budget and time allowance specified. I look forward to hearing your input on the proposed solution. Please let me know if you have any questions.

Sincerely,

Katrina Stradford

Katrina Stradford
Senior Machine Learning Engineer

Part B: Project Proposal Plan

Project Summary

Alaska University (AU) is a public, four-year university that offers both undergraduate and graduate degree programs. Over the past three years, AU has experienced an 85% increase in graduate applicants. During this time, an increasing number of admitted students have declined enrollment because of the university's delayed admissions decisions. In addition, the university's applicant satisfaction rate has decreased by 20% over the last two years.

To maintain AU's ability to enroll top students in its educational programs, the organization aims to automate its admissions decision processes. To do so, the university plans to deploy a machine learning model that categorizes applicants with an admissions decision and assigns a probability percentage representing the applicant's likelihood of admission. This switch to automation rather than a fully manual application review is proposed to remedy AU's problem with late admissions decisions. The machine learning solution uses a logistic regression model to accept a dataset of Alaska University applicants and output admissions decisions for each applicant. The solution can also render an admissions decision or likelihood of admission for an individual applicant.

The solution will be presented via the Admissions Decision Machine Learning Integration Tool (ADMIT). ADMIT is an internet dashboard where admissions department personnel can render admissions decisions and see visual representations of applicant data. The dashboard will be accompanied by a user guide, which contains step-by-step tutorials on how to use ADMIT's features. Alaska University's current decline in ability to enroll quality students and maintain applicant satisfaction is largely because of a time-consuming admissions process. ADMIT will offer the benefit of reducing the manual admissions decision process to minutes. In addition, the probability of admissions offered for individual applicant entries can reduce additional time spent reviewing admission decision appeals.

Data Summary

The data needed to create the machine learning application is available at <https://www.kaggle.com/datasets/sakshisaku3000/admission-predict>. This data represents a sample of Alaska University's applicants over the past three years and will be downloaded in CSV format. Data used within the live application after deployment should be collected from the university's applicant tracking system (ATS). Once the application renders the admissions decisions the user will be able to re-download the CSV file of applicants. This updated CSV file can be uploaded to the university's existing student information system (SIS) to continue the admissions and enrollment processes.

Each record of the data file represents one applicant and contains that applicant's Graduate Readiness Exam (GRE) score, cumulative grade point average (CGPA), statement of purpose score (SOP), and a boolean value representing whether the student was admitted or rejected. The CGPA and GRE scores are entered by the student during the application process and later verified before enrollment. The SOP is manually assigned by admissions faculty and input into the ATS before the data is exported and transferred to ADMIT.

While GPA is a standard in admissions determination, including application criteria that show a student's level of interest is also important for student retention and success (Felton, 2015). Since Alaska University's current admissions process already considers GRE and SOP scores, these scores, along with applicant CGPA, can determine admissions decisions without heavily disrupting the current admissions process.

Alaska University's student application does not allow applications to be submitted with missing data fields. Therefore, data completeness is ensured. To ensure data accuracy, the exported CSV file of applicants should be explored to confirm all scores are within an acceptable range. To ensure FERPA and GDPR compliance, all PII should be excluded from the ATS data export.

Before the data is used to train the machine learning model, it will be cleaned and preprocessed. This includes exploratory data analysis (EDA). During EDA, the analyst will ensure accuracy and check for outliers, missing data, and duplicates. During model development, the data will be split into training and testing sets and used to train the model. During evaluation, additional data will be generated from <https://mockaroo.com> to test the demo application. Once the application is deployed, logged-in users can upload and download the data and accompanying admissions decisions using the ADMIT dashboard.

Implementation

The six CRISP-DM steps below will guide the development of the machine learning model:

Business Understanding

The Project Manager reviews business needs, project requirements, and data availability with clients and shareholders. The Project Manager will then complete and send the project's proposal, which includes data requirements.

Data Understanding

The Data Scientist will source the required data, perform exploratory data analysis, and confirm the data quality.

Data Preparation

The Data Scientist cleans the data and selects features and label(s). The data is checked for outliers and accuracy.

Modeling

The Machine Learning Engineer chooses a machine learning algorithm, splits the data into training and testing sets, and trains the model using the training dataset. The model is tested using the testing set, and the MLE verifies that the model renders admissions decisions.

Evaluation

The Software Engineer builds and deploys the demo web application that includes the machine learning model built during the modeling phase. The QA Engineer generates the demo dataset and tests the model's functionality within the web application.

Deployment

The Software Engineer deploys the web application to the university's hosting platform. The QA Engineer verifies that the live web application meets the project's requirements. The Software Engineer, Machine Learning Engineer, and QA Engineer create a monitoring and maintenance plan and the User Guide.

Timeline

Milestone	Duration	Projected Start	Projected End
Business Understanding Verify project requirements, confirm data requirements, and create project proposal	1 week	07/29/2024	08/02/2024
Data Understanding Find data source, query and visualize data, and confirm data quality	1 week	08/05/2024	08/09/2024
Data Preparation Clean data, select features, and format data	2 weeks	08/12/2024	08/23/2024

Modeling Select algorithm, split data, build the model, validate the model	1 week	08/26/2024	08/30/2024
Evaluation Build and deploy demo application, test model within demo application	1 week	09/02/2024	09/06/2024
Deployment Deploy final web application, verify final application, plan monitoring and maintenance, create the user guide	3 weeks	09/09/2024	09/27/2024
TOTAL TIMELINE	9 weeks		

Table 1: projected project schedule

Evaluation Plan

The product will be evaluated during each phase of development, as detailed below:

Business Understanding

The Dean of Admissions verifies that the project proposal meets the admissions department's needs and goals and provides approval before the project begins.

Data Understanding

The Project Manager verifies the dataset presented by the Data Scientist meets the requirements listed in the project proposal.

Data Preparation

The Data Scientist and Machine Learning Engineer verify data accuracy, validity, and cleanliness. The Project Manager verifies the features selected are in alignment with the project's requirements.

Modeling

The Machine Learning Engineer verifies the model has been trained and tested. The Machine Learning Engineer validates the model using accuracy, f-1 score, and AUC score.

Evaluation

The Machine Learning Engineer and Project Manager review the model's validation results. The Software Engineer performs unit testing to verify the application's functionality. The QA Engineer verifies that the demo application incorporates the machine learning model effectively and meets the requirements listed in the project proposal.

Deployment

The Software Engineer verifies that the application has been successfully migrated to the university's faculty web portal. The QA Engineer verifies that the live web application and user guide meet the project's requirements. The Project Manager verifies that the monitoring and maintenance plan fits the client's expectations. The Dean of Admissions validates the model's ability to provide the Spring 2025 list of admitted students. Several admissions department staff members audit the system to ensure user-friendliness.

Resources and Costs

Resource	Description	Cost
Project Manager	Rate: \$60/hour 30 hours a week for 9 weeks	\$16,200
Machine Learning Engineer	Rate: \$62/hour 30 hours a week for 9 weeks	\$16,740
Data Scientist	Rate: \$45/hour 30 hours a week for 4 weeks	\$5,400
Software Engineer	Rate: \$40/hour 30 hours a week for 3 weeks	\$3,600
QA Engineer	Rate: \$47/hour 30 hours a week for 3 weeks	\$4,230
Hardware	personal laptop for each contractor	\$5,000
University Application Portal	existing application portal and applicant tracking system	\$0

Fly.io	demo application hosting	\$5/month for 1 month
University Faculty Web Portal	live application hosting	\$0
Python, NumPy, Scikit-learn, Pandas	Python programming language and its associated libraries	\$0
Data	sourced data, university-owned data, and generated data	\$0
	Total	\$51,175

Table 2: itemized list of costs

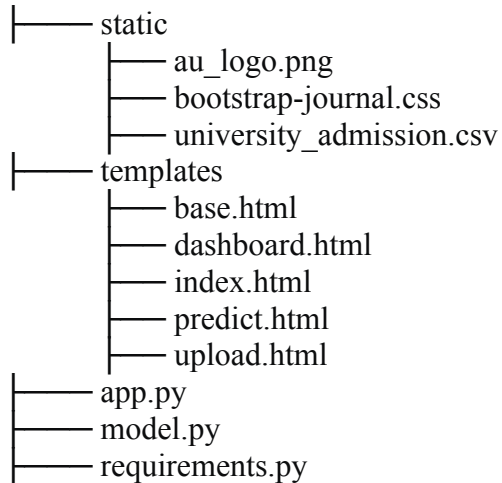
Part C: Application

Application Link

The application can be accessed at <https://alaskaadmissions.fly.dev/>.

Application Files

AlaskaAdmissions



Data Files

au_admissions.csv - data file that is saved after cleaning, formatting, and adding applicant no.

demo_applicants.csv - mockaroo.com generated data used to test the demo user interface

university_admission.csv - raw data file

Part D: Post-Implementation Report

Solution Summary

The Alaska University admissions department was seeking an application that would help manage the growing applicant pool and reduce the time spent reviewing admissions applications. The **Admissions Decision Machine Learning Integration Tool (ADMIT)** allows up to 5000 admissions decisions to be generated within minutes.

The application classifies applicants as eligible or ineligible for admission using a supervised machine learning model that was trained using previous Alaska University admissions decisions. ADMIT uses a logistic regression algorithm to offer a prescriptive sidebar that can render bulk or individual admissions decisions and a descriptive main dashboard that contains data plots and graphs.

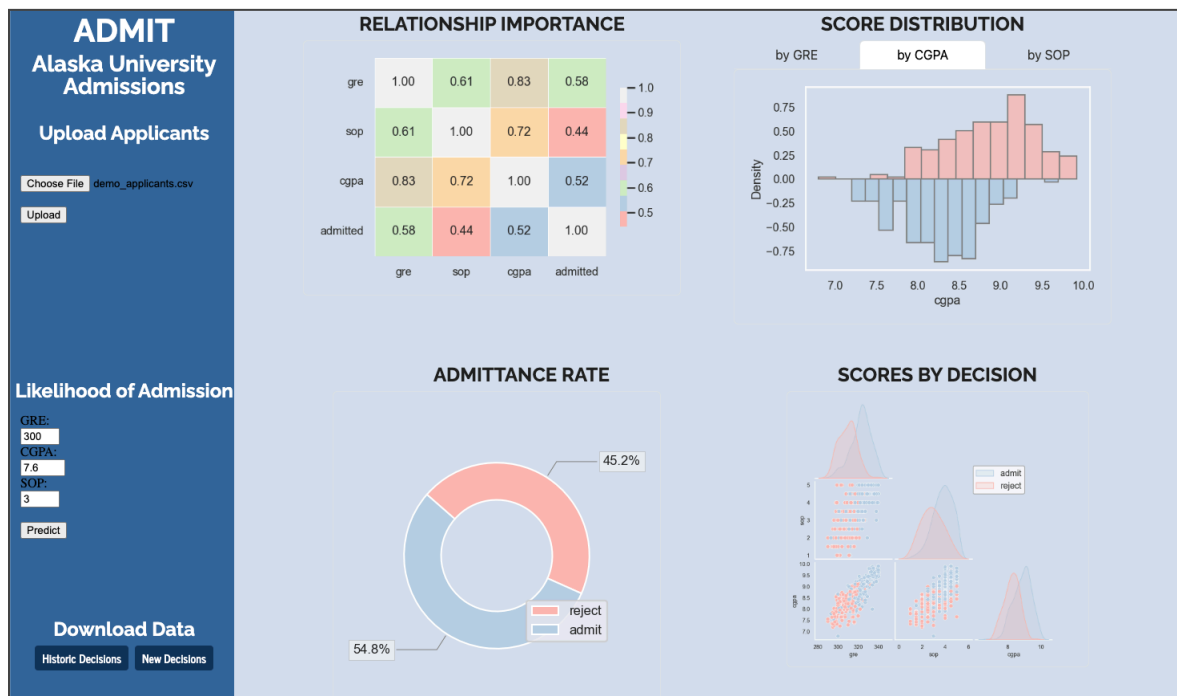


Figure 1: ADMIT dashboard

Data Summary

The data used to train the model can be found at <https://www.kaggle.com/datasets/sakshisaku3000/admission-predict>. It is a representative sample of Alaska University's admissions decisions over the past three years. The raw data contained 400 records, each representing an applicant. The data available for each applicant includes their

Graduate Record Examination (GRE) score, statement of purpose (SOP) score, cumulative grade point average (CGPA), and their admissions decision ([Figure 2](#)).

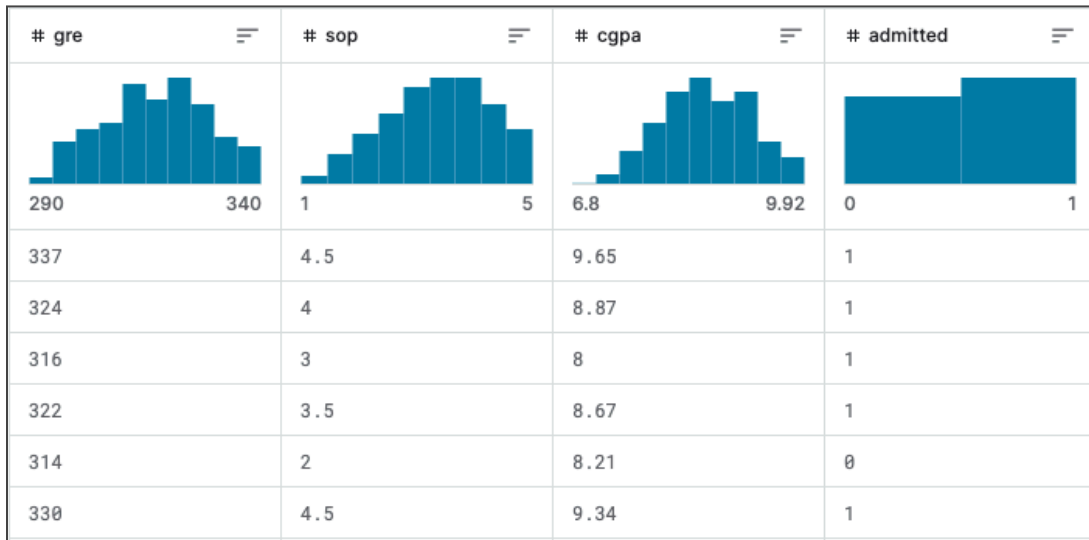


Figure 2: raw data sample

During the data understanding phase of development, the Data Scientist imported the data and performed exploratory data analysis (EDA). This process began with the raw CSV data file being imported as a dataframe using the Pandas library. It was confirmed that there was no PII within the data. During EDA, the `isnull()` and `describe()` methods were called. When using `.sum()` with `isnull()`, the output is a table showing the number of null values present in each data column. Our data was free from null values, which was expected since the application automatically does not accept applications with missing values. The `describe` method returns information about the value distribution and size of each column ([Figure 3](#)).

	applicant no.	gre	sop	cgpa	admitted
count	400.000000	400.000000	400.000000	400.000000	400.000000
mean	200.500000	316.807500	3.400000	8.598925	0.547500
std	115.614301	11.473646	1.006869	0.596317	0.498362
min	1.000000	290.000000	1.000000	6.800000	0.000000
25%	100.750000	308.000000	2.500000	8.170000	0.000000
50%	200.500000	317.000000	3.500000	8.610000	1.000000
75%	300.250000	325.000000	4.000000	9.062500	1.000000
max	400.000000	340.000000	5.000000	9.920000	1.000000

Figure 3: snippet of `df.describe()` output

When analyzing the output from `describe()`, we saw that the CGPA column's minimum value was over 2 standard deviations (std) below its mean, which is our benchmark for an outlier. To get a better view of data distribution, Seaborn boxplots were created for each feature ([Figure 4](#)). The CGPA data point outside of the boxplot's whisker area shows the outlier. However, this data

point was kept in the data set because of the nature of the data. In future use of the application, all applicants must receive an admissions decision, even when scoring outside of the normal range. Instead of removing the data, outliers were handled later by scaling the data before fitting the machine learning model. Applications with duplicate PII are automatically merged within the application portal before the applicant data is exported so no applicant is represented twice. We did not check for other duplicates since multiple applicants could have the same scores. To ensure data quality and accuracy, we also confirmed that each score column contained scores within an appropriate range. These ranges are 260-340 for GRE scores, 1-5 for SOP scores, and 0.5-10.0 for CGPA scores.

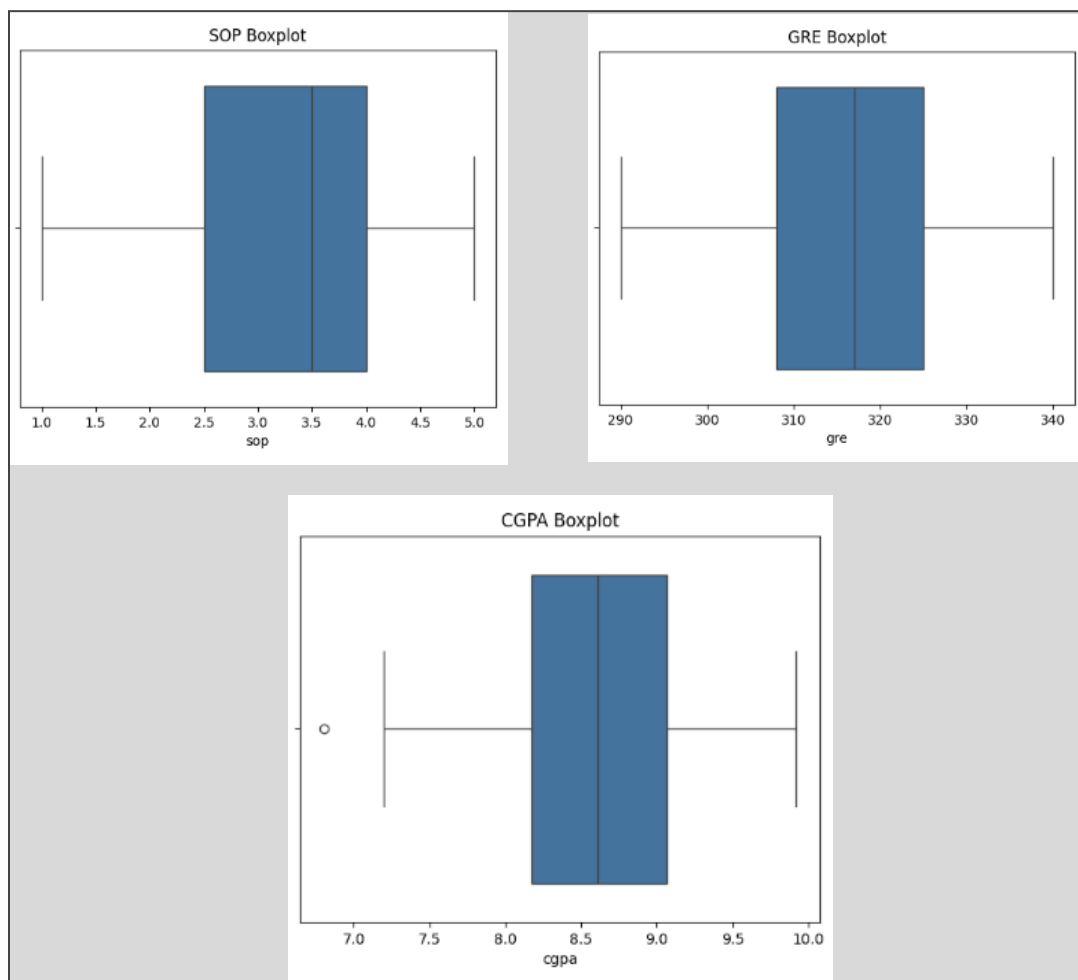


Figure 4: feature boxplots

During data preparation, it was confirmed that the data was fully cleaned and features were selected. To make the data suitable for the application's final use, applicant numbers were re-added to the dataframe starting at 1 and incrementing up to the length of the dataframe. This will already be present in the data when exporting from the applicant tracking system. The

logistic regression algorithm requires boolean values 0 and 1 for the label, so we kept the admitted column in the boolean format. Per standard boolean logic, 0=false and 1=true. This means that applicants with a 0 in this column were rejected and applicants with a 1 were admitted. The formatted dataframe was saved as a CSV file under the name *au_admissions.csv* in the project's directory. The code used to complete these steps can be seen in [Figure 5](#) and the resulting dataframe can be seen in [Figure 6](#).

```
# %%  
# import data and print dataframe  
data = "./static/university_admission.csv"  
df = pd.read_csv(data)  
df  
  
# %%  
df.isnull().sum()  
  
###  
det = df.describe()  
det  
  
# %%  
# gre boxplot  
fig, ax = plt.subplots()  
gre_box = sns.boxplot(x=df["gre"])  
plt.title("GRE Boxplot")  
plt.show()  
plt.close()  
  
# sop boxplot  
sop_box = sns.boxplot(x=df["sop"])  
plt.title("SOP Boxplot")  
plt.show()  
plt.close()  
  
# cgpa boxplot  
cgpa_box = sns.boxplot(x=df["cgpa"])  
plt.title("CGPA Boxplot")  
plt.show()  
plt.close()  
  
# %%  
# insert applicant no.  
df.insert(loc=0, column='applicant no.', value=np.arange(1, len(df) + 1))  
  
# %%  
# save new dataframe to csv for dashboard  
df.to_csv(path_or_buf="./static/au_admissions.csv", index=False)
```

Figure 5: EDA and data formatting

Data View df						
	123 applicant no.	123 gre	123 sop	123 cgpa	123 admitted	
0	1	337	4.5	9.65	1	
1	2	324	4.0	8.87	1	
2	3	316	3.0	8.00	1	
3	4	322	3.5	8.67	1	
4	5	314	2.0	8.21	0	
5	6	330	4.5	9.34	1	
6	7	321	3.0	8.20	1	
7	8	308	3.0	7.90	0	
8	9	302	2.0	8.00	0	
9	10	323	3.5	8.60	0	
10	11	325	3.5	8.40	1	

Figure 6: formatted data

For demoing purposes, additional mock applicant data was generated using <https://mockaroo.com>. This data contains the features but not the label as the application inputs the admissions decisions during the upload process. Implementation of this process is shown in Figure 11 in the Machine Learning section of this document. A portion of the data rendered at mackaroo.com is shown below in Figure 7.

demo_applicants			
applicant no.	LOR	SOP	CGPA
1	272	1	6.6
2	290	3	6.4
3	318	4	8.0
4	339	1	8.4
5	297	1	8.0
6	297	4	7.2
7	269	1	6.3
8	272	4	6.8
9	298	4	8.7
10	323	3	7.9

Figure 7: demo data snippet

Machine Learning

The ADMIT dashboard's ability to render admissions decisions and the likelihood of admission given applicant scores is made possible by its binary logistic regression model. Logistic regression is a supervised learning algorithm that assumes a linear relationship between each feature and the label. This assumption is used to render a sigmoid curve on the domain $[0,1]$, which signifies the probability of the label event occurring. Classifications are made when features are mapped to a location on the curve and compared against an acceptance threshold.

Receiving both classifications and the probability of applicant admission will expedite admissions decisions and also provide valuable information for review processes like admissions appeals. Logistic regression models are also computationally light while scaling (Sagir, n.d.), which can be an asset when building a model that will need to expand with a growing applicant pool.

The logistic regression implementation was done with the Scikit-learn Python library. Within Scikit-learn, both the `LogisticRegression` and `LogisticRegressionCV` classes are available. The difference between these two classes is that `LogisticRegressionCV` automatically implements cross-validation with other optimal hyperparameters. Both versions were tested and ultimately the `LogisticRegressionCV` class resulted in better results. With this selection, the lbfgs solver, ℓ_2 regularization, and stratified k-fold validation are automatically chosen as hyperparameters. Since the dataset being used to train the model is relatively small, protection against overfitting via these hyperparameters is likely the cause of this class performing better than the standard `LogisticRegression` class. These parameters also minimize log loss, which is the difference between the model's predicted probabilities and the actual probabilities.

Before ℓ_2 regularization could be applied, we accounted for the outlier we found in the CGPA data. We did this by splitting the independent and dependent variables into separate dataframes and scaling the independent variables using Scikit-learn's standard scaler ([Figure 8](#)). [Figure 9](#) below shows the independent data before and after the scaling process. The scaled independent variables, X , were then divided into an 80/20 train/test split. The dependent, y , variables were split in the same manner. The logistic regression model was then trained using the X and y training sets. To optimize the model's performance and create duplicatable results, the following parameters were manually set when training the model: 5 cross-validation folds, a random state of 42, and `refit` set to `True`.

```

# %%
# group independent vs. dependent variables
X = df.drop(columns=['applicant no.', 'admitted'])
y = pd.DataFrame(df['admitted'])

# %%
# scale features
sc = StandardScaler()
X = sc.fit_transform(X)

# %%
# split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(*arrays: X, y, test_size=0.2)

# %%
# train model
y_train_array, y_test_array = y_train['admitted'].values, y_test['admitted'].values
log_model = LogisticRegressionCV(cv=5, random_state=42, refit=True).fit(X_train, y_train_array)

```

Figure 8: preparing for and training model

X Before Scaling			X After Scaling		
gre	sop	cgpa	0	1	2
337	4.5	9.65	1.762107	1.093864	1.764818
324	4.0	8.87	0.627656	0.596653	0.455151
316	3.0	8.00	-0.070467	-0.397769	-1.005631
322	3.5	8.67	0.453126	0.099442	0.119339
314	2.0	8.21	-0.244998	-1.392191	-0.653029
330	4.5	9.34	1.151249	1.093864	1.244310
321	3.0	8.20	0.365860	-0.397769	-0.669819
308	3.0	7.90	-0.768590	-0.397769	-1.173537
302	2.0	8.00	-1.292182	-1.392191	-1.005631
323	3.5	8.60	0.540391	0.099442	0.001805
325	3.5	8.40	0.714922	0.099442	-0.334007

Figure 9: independent variables before and after scaling

After fitting the model, we used the built-in `predict_proba()` and `predict()` functions to get the probability of each applicant in the test set being admitted as well as their resulting classification. The built-in threshold for the `LogisticRegressionCV` class is 0.5. Each applicant is categorized with either 0 or 1 in the `admitted` column, based on if their probability prediction is at or above 0.5 in the 0 or 1 column of the probabilities table. For example, the first applicant in the

Probabilities table received a rounded prediction of 76% probability of 0 (or rejection) and 24% probability of admission. Therefore, when we look at this applicant in the Predictions table, their predicted classification is 0 since that probability is at or above 0.5, or 50%.

Probabilities				Predictions			
∇	\div	$\overline{123} \ 0 \ \nabla$	\div	$\overline{123} \ 1 \ \nabla$	\div	∇	\div
0		0.760691		0.239309		0	0
1		0.707884		0.292116		1	0
2		0.081924		0.918076		2	1
3		0.145698		0.854302		3	1
4		0.187791		0.812209		4	1
5		0.153559		0.846441		5	1
6		0.201254		0.798746		6	1
7		0.675510		0.324490		7	0
8		0.130545		0.869455		8	1
9		0.341403		0.658597		9	1
10		0.061558		0.938442		10	1

Figure 10: model probabilities vs. predictions

Within the ADMIT dashboard, predictions are used in the Upload Applicants area and probabilities are recalculated and used when the Get Likelihood of Admission form is filled out. Figures 11 and 12 below showcase how these two functions were incorporated into the application. These two methods, and the application, were designed using the Flask framework.

```
@app.route(rule: '/upload', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # retrieve uploaded file, create dataframe, and scale ind. variables
        file = request.files['applicants']
        new_data = pd.read_csv(file)
        X = new_data.drop(columns=['applicant no.'])
        X = pd.DataFrame(model.sc.transform(X.values))

        # apply ml model and insert admitted into dataframe
        decisions = model.log_model.predict(X)
        new_data.insert(loc=4, column='admitted', decisions)

        # save new data to csv and display dashboard
        new_data.to_csv(path_or_buf='static/new_data.csv', index=False)
        dashboard()

        return render_template('upload.html')
    else:
        return render_template('upload.html')
```

Figure 11: submitting the Upload Applicants form calls the upload function in app.py, which gets the admissions decisions and saves the file

```

@app.route(rule: '/predict', methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
        # retrieve and scale scores from the form
        gre = float(request.form.get('gre'))
        cgpa = float(request.form.get('cgpa'))
        sop = float(request.form.get('sop'))
        scores = (gre, cgpa, sop)
        sc_scores = model.sc.transform([scores])

        # generate prediction and probability using ml model
        prediction = model.log_model.predict(sc_scores)
        if prediction[0] == 0:
            prediction = 'Reject'
        else:
            prediction = 'Admit'
        probability = model.log_model.predict_proba(sc_scores)

        return (f'{probability[0][1] * 100:.2f}% Chance of Admission'
                f' '
                f'Recommendation: {prediction}')

    else:
        return render_template('predict.html')

```

Figure 12: submitting the Get Likelihood of Admission form calls the predict function in app.py, which returns the probability score and prediction

Validation

The Scikit-learn methods used to validate the model were accuracy, f-1 score, and auc score. The score required for the model to be validated as successful was set at 0.75 or 75% for each metric. Validation of the model was confirmed since all three scores were above 0.75 (Figure 13).

While accuracy is a popular metric, it isn't always the most reliable for admissions models. Most universities have an exponentially higher probability of rejecting an applicant than they do of admitting them. Therefore, if a machine learning solution returned rejections for every candidate, the accuracy rate would be high, but this may not reflect the model's true ability to predict. The students at the University of Texas made this discovery after their baseline model received an accuracy score of 84.4% by rejecting every applicant (Waters & Miikkulainen, 2014).

<pre> # %% # get scores accuracy = accuracy_score(y_test, predictions) f1 = f1_score(y_test, predictions) auc_score = roc_auc_score(y_test, predictions) # print scores with two decimal places print(f"Accuracy: {accuracy:.2f}") print(f"F1 Score: {f1:.2f}") print(f"AUC Score: {auc_score:.2f}") </pre>	<p>Accuracy: 0.81</p> <p>F1 Score: 0.82</p> <p>AUC Score: 0.82</p>
--	--

Figure 13: score breakdown

To visualize the model's accuracy, we plotted a confusion matrix using the Scikit-learn and Matplotlib libraries (Figure 14). The confusion matrix shows 38 true positives, 12 false positives, 26 true negatives, and 4 false negatives. In this circumstance, true positives (TP) are applicants who were correctly predicted as admitted. False positives (FP) refer to rejected applicants whom the model predicted as admitted. True negatives (TN) were applicants who were accurately predicted rejected. False negatives (FN) refer to applicants who should have been predicted as admitted but were instead predicted as rejected. True positives and true negatives the ideal prediction scenarios. False positives have the potential to jeopardize student success and retention, as we would admit students who were not fit to attend AU. False negatives have the potential to lower our applicant satisfaction rate and to increase the time the admissions staff takes on application appeals processes.

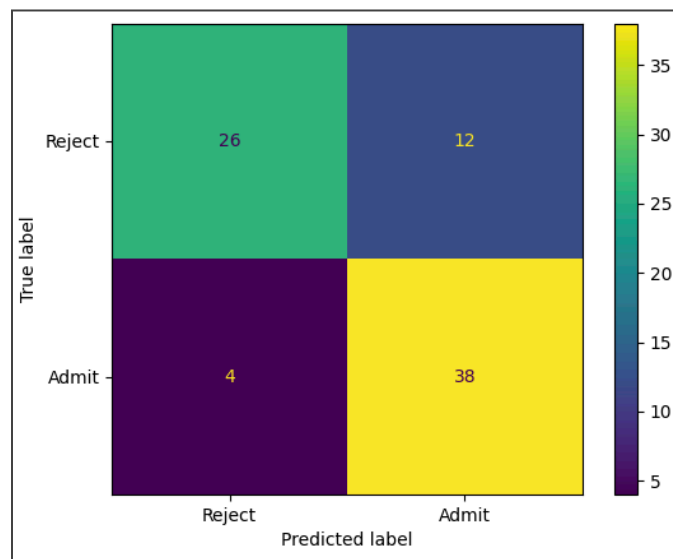


Figure 14: Confusion Matrix

Accuracy

Accuracy is measured as $\frac{\text{true positives} + \text{true negatives}}{\text{all predictions}}$ and is an indicator of how often the model's predictions are correct. With 81% accuracy and a near 50/50 split between admitted and rejected applicants (Figure 18), our model has proven its ability to categorize students appropriately for both admission and rejection.

F-1 Score

The f-1 score is a blend of precision (correctly identifying qualified applicants) and recall (discerning between the right and wrong applicants to admit). It measures the model's ability to make accurate positive predictions while avoiding inaccurate predictions. An f-1 score of 0.82

shows that the model can compare applicants and categorize them appropriately against each other.

AUC Score

The ROC curve (Figure 15) measures accurately admitted applicants (TP) against wrongfully admitted applicants (FP) at different thresholds. The area under the curve (AUC) score illustrates the model's ability to predict one applicant's standing when compared to another. Our AUC score of 0.82 confirms that the model can effectively rank applicants against one another.

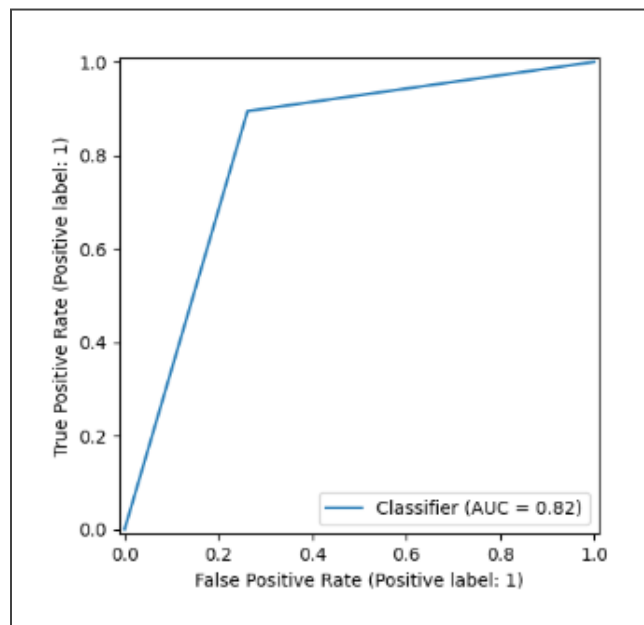


Figure 15: ROC Curve

Efficiency

The application's ability to cut 50% of the admissions process time will be validated by comparing the Spring 2025 semester time cards to the timecards for the prior three semesters.

Applicant Satisfaction

The application's ability to raise applicant satisfaction by 20% will be assessed by reviewing and tallying applicant satisfaction surveys at the end of the Spring 2025 semester. The average score will be compared to the average of the prior three semesters.

Visualizations

The ADMIT dashboard contains four descriptive visual elements to guide the user in assessing the data. When the dashboard initially loads, these visuals are populated based on AU's historic

admissions data, which is the same data used to train the model. After new data is entered using the Upload Applicants form, the charts will automatically refresh to represent the new data (Figure 11). The graphs in this section were created using the Matplotlib and Seaborn libraries. The NumPy library was also used for mathematical calculations. The user dashboard contains the following visual guides:

Scatterplot Matrix

This visual shows the linear relationship between variables and a brief distribution of scores segmented by the admission decision. Straighter vertical or horizontal plots show weaker correlation between variables, while diagonal plots show stronger correlation between them. In the matrix below, created using Alaska University's historical data, we can see a strong correlation between GRE and CGPA scores. SOP scores, however, are not highly correlated to either GRE or CGPA scores. If the admissions department were looking to simplify the application process and require fewer scores from students, SOP scores could be best to exclude. This could also create an even more productive admission process for AU, as scoring SOP scores is the only manual process that is still required after ADMIT's implementation.



Figure 16: created with Matplotlib library

Correlation Heatmap

The correlation heatmap shows how strongly each variable influences the others and which score(s) have the most influence on admittance. Higher scores show a stronger correlation, and lower scores show more independence between the two compared variables. In some cases the scores can be negative numbers. In that case, the absolute value of the score is assessed as its value.

The correlation heatmap can be used by admissions staff to tailor their application requirements or to change the model's features in the future. In our historic heatmap below, we see the same patterns of relevance and irrelevance that we saw in the scatterplot matrix. The highest

intersectional pair was GRE and CGPA at 0.83, confirming their strong connection. When we look at the Admitted row, we also see confirmation that SOP scores are less strongly correlated to final admissions decisions.

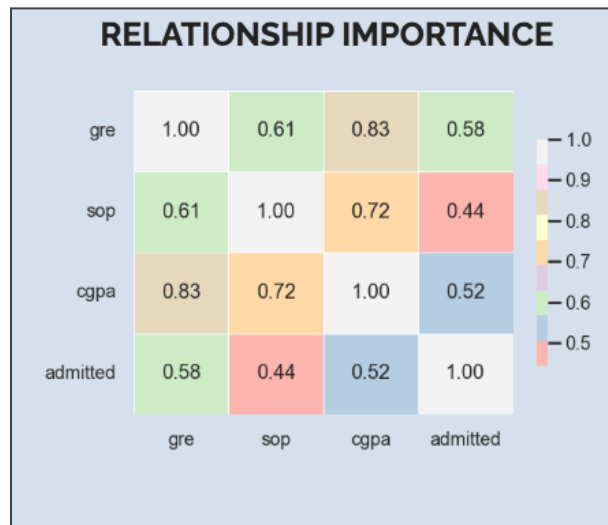


Figure 17: created with Seaborn and Matplotlib libraries

Pie Chart

The pie chart shows the admittance and rejection rate (Figure 18). The percentage of rejected applicants is shown in red, and the percentage of admitted applicants is shown in blue. Applicants are admitted or rejected based on merit, so a higher percentage of admitted applicants represents a higher quality applicant pool and vice versa. This information could be useful in knowing when to focus on admissions marketing efforts and understanding the university's reputation among prospective applicants.

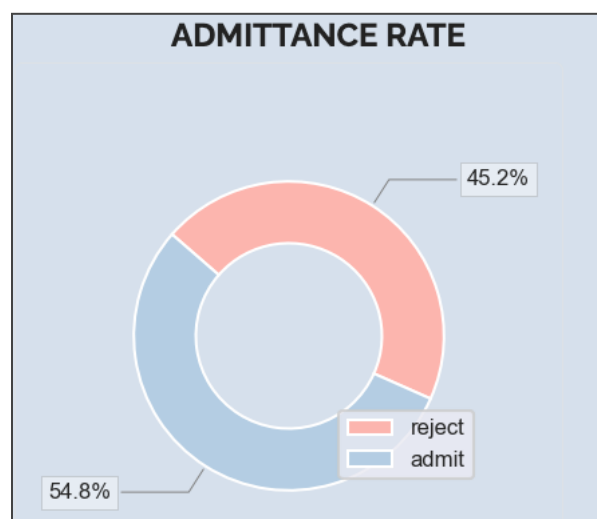


Figure 18: created with Matplotlib library

Histograms

There are three histograms available. They are grouped by GRE scores, SOP scores, and CGPA scores. Each histogram shows the distribution of scores amongst admitted applicants vs. rejected applicants. In the future, more histograms could be added to segment applicants by their prospective degree program. Heavily skewed histograms will show that a score is a strong determinant of acceptance or rejection. Evenly distributed scores will show less importance. Admissions staff can use the histograms to determine minimum score requirements over time as the need to become more selective arises. In the historic data histogram below, the accepted values skew heavily to the left, while the rejected values have a more symmetrical distribution. This tells us that a higher CGPA can be a factor in being offered admission, but there is likely another score or a combination of scores that are a more reliable indicator. We also see that it's rare for an applicant to be rejected with a CGPA above 9.0 and just as rare for an applicant to be admitted with a CGPA below 8.0.

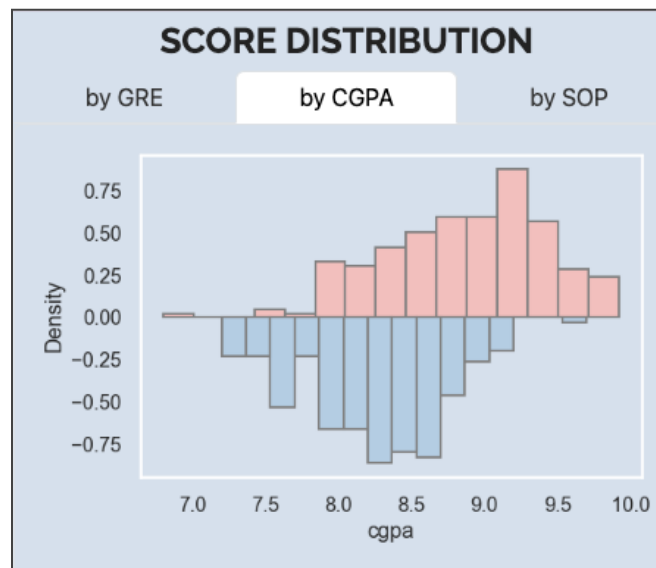


Figure 19: created with Seaborn and Matplotlib libraries

User Guide

During the demo period, please allow up some time for the demo application to load and render chart updates. This latency will be minimized once the application is fully deployed to Alaska University's existing employee web portal.

Accessing the Application

1. Open a web browser and navigate to <https://alaskaadmissions.fly.dev>.

2. Log in using your provided username and password. Click Submit. Note: Usernames and passwords are case-sensitive.

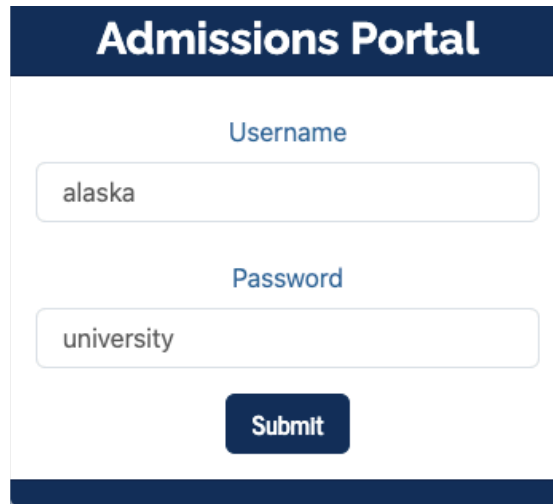
A login form titled "Admissions Portal" in a dark blue header. Below the header, there are two input fields. The first is labeled "Username" in blue text and contains the text "alaska". The second is labeled "Password" in blue text and contains the text "university". Below these fields is a dark blue button with the word "Submit" in white text.

Figure 20: enter username and password and click Submit to enter the dashboard

Uploading Applicants

This feature requires a CSV file of applicant scores with the columns applicant no., gre, sop, and cgpa.

1. Locate the **Upload Applicants** section in the sidebar.
2. Add the new applicant file by clicking “Choose File” and selecting the file from your computer’s file directory. **The demo_applicants.csv file provided can be used to demo this feature.** You may also drag and drop files into this area. When successful, the “No file chosen” text will change to show the filename ([Figure 21](#)). Note: Data files with previously filled admissions decisions can not be uploaded in this area.
3. Click Upload once your file has been added. The dashboard’s charts will update to reflect the new data. A brief delay is normal before the charts regenerate.

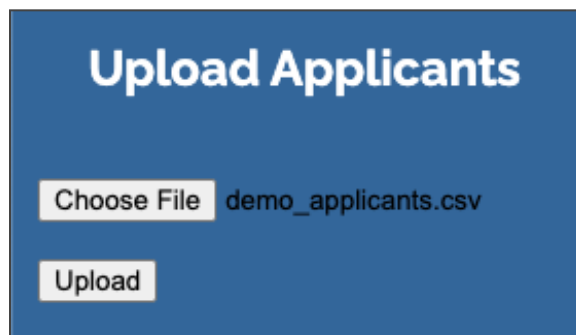
A form titled "Upload Applicants" in a dark blue header. Below the header, there is a "Choose File" button followed by the text "demo_applicants.csv". Below this is an "Upload" button.

Figure 21: upload form with the demo file selected

Downloading Admissions Decisions

Historic Decisions

This button downloads a copy of the historical data used to train the machine learning model and to populate the initial charts when you log in to the portal.

New Decisions

This button re-downloads the new applicant file after adding the admissions decisions. New data must be uploaded before using this button.



Figure 22: download historic or new admissions decisions

Get Individual Admissions Decisions

1. Locate the **Likelihood of Admission** section on the sidebar.
2. Enter the applicant's GRE, CGPA, and SOP scores.
3. Click "Predict" to generate an admissions decision and a percentage representing the probability that the applicant would be admitted based on historic admissions decisions.

Input Requirements:

GRE: must be an integer between 260-340. Ex: 270

CGPA: must be a single decimal point number between 0.5-10.0. Ex: 8.7

SOP: must be a single decimal point number in increments of 0.5 between 0.5-5.0. Ex: 3.5

A blue rectangular form titled 'Likelihood of Admission'. It contains three input fields: 'GRE:' with the value '300', 'CGPA:' with the value '7.6', and 'SOP:' with the value '3'. Below these fields is a 'Predict' button.

Figure 23: predict form with sample data

References

- Felton, E. (2015, August 21). *Colleges shift to using “big data” — including from social media — in admissions decisions*. The Hechinger Report.
https://hechingerreport.org/colleges-shift-to-using-big-data-including-from-social-media-in-admissions-decisions/?utm_source=Sailthru&utm_medium=email&utm_campaign=Issue:%202016-03-02%20Higher%20Ed%20Education%20Dive%20Newsletter%20%5BIssue:5106%5D&utm_term=Education%20Dive:%20Higher%20Ed
- Sagir, U. (n.d.). *Logistic Regression Pros & Cons*. HolyPython.com.
<https://holypython.com/log-reg/logistic-regression-pros-cons/>
- Waters, A., & Miikkulainen, R. (2014). GRADE: Machine Learning Support for Graduate Admissions. In *Association for the Advancement of Artificial Intelligence*. AI Magazine.
<https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2504>