# H2B Visa Outcome Analysis and Prediction

ST310 Group Project

Candidate Numbers: 46524, 48605, 45016, 45686

## Introduction

The H-1B visa programme allows US employers to hire foreign professionals for specialised roles such as engineers, scientists and software developers. Although the US Citizenship and Immigration Services (USCIS) uses an electronic registration system, the H-1B visa approval process remains predominantly manual, involving extensive paperwork, and processing times ranging from approximately 2.5 to 8 months per case (Florida Tech; USCIS). Moreover, whenever there is a human element in such a process, there is always a risk of subjectivity, which can potentially lead to unfair decision-making. This labour-intensive and complex system can sometimes lead to inefficiencies and inconsistencies, highlighting the potential for automation and machine learning driven decision-making algorithms.

Building upon this premise, this project investigates how machine learning models could improve the H-1B visa approval process. Specifically, how machine learning algorithms can automate preliminary decision-making by predicting visa status outcomes based on historical data. This can reduce workload, accelerate processing times, and promote objective evaluations. Additionally, the dataset used in this project excludes potentially discriminatory attributes (e.g., age, gender, or race), which further ensures fairness. Beyond automation, these models can also uncover critical insights into the factors driving visa decisions.

The core objective of this project is to build several machine learning models that classify visa cases as either "Confirmed" or "Denied". Furthermore, through these predictive models, we will also analyse which factors are most significant in determining visa outcomes and whether there are meaningful interactions between predictors. Additionally, since some machine learning models may function as "black boxes," we will dedicate a section of our analysis to developing an interpretable model. This approach will provide transparency and clarity regarding how predictions are derived, ensuring the decision-making process remains understandable to stakeholders.

Our analysis uses the "H1B Non-immigrant Labour Visa" dataset sourced from Kaggle, containing over 3 million visa application records filed between 2011 and 2018. The dataset includes 23 predictors on features such as employer name, job specifications, wage details, and geographic location data. The output variable is *case_status*, which is a categorical variable with levels "Confirmed", "Denied", "Confirmed-Withdrawn", and "Withdrawn".

## Methods

This section outlines the approach used to preprocess the data and details the five machine learning models aimed at predicting H-1B visa petition outcomes ("Confirmed" or "Denied").

### Preprocessing

The preprocessing pipeline involved several critical steps to prepare the H-1B visa petition data for model building. All transformations were learned from the training dataset and subsequently applied to the validation and test datasets to avoid any data leakage.

1. Feature Removal

The following columns were explicitly removed from all datasets:

- *decision_date*: Removed due to potential target leakage, as this date is unavailable at the time of petition evaluation.
- *emp_country*: Removed due to lack of predictive value (99.9% of observations listed as "USA").
- *wage_to* and *pw_level*: Removed due to high proportion of missing values (>50%) and redundancy with existing wage-related columns.

2. Missing Outcome Values

We removed rows with missing outcome values (*case_status*). Only less than 5 rows were missing.

3. Recipe-based Transformation Pipeline (tidymodels)

Using tidymodels, a transformation pipeline (recipe) was implemented, including the following steps:

- **Nominal predictors**: Converted to categorical factors
- **Date features** (*case_submitted*): Transformed into meaningful temporal components: year (*case_year*), month (*case_month*), and weekday (*case_wday*). Original date feature (*case_submitted*) was subsequently removed (step_rm)
- **Rare categories**: Consolidated into "other" groups if frequency was below 1%
- **Textual fields** (*emp_name*, *job_title*, *soc_name*): Standardised naming conventions using fct_relabel to ensure R-compatible category labels and collapsed less frequent levels into "other" category (only kept the top 50 most frequent categories)
- **Highly missing categorical columns** (*emp_h1b_dependent*, *emp_willful_violator*, *full_time_position*): Missing values explicitly coded as "unknown"
- **Missing numeric predictors**: Filled using median imputation
- **Novel categorical levels**: Handled by introducing a new "novel" category
- **Missing nominal predictors**: Filled using mode imputation
- **Numeric predictors**: Normalised (scaled and centred) to standardise their distributions
- **Multicollinearity**: Highly correlated numeric predictors (correlation $> 0.9$) were removed to reduce redundancy (none of the columns were removed)

First, we set a reproducible random seed (46524) and loaded the full H-1B dataset. After standardising the outcome by recoding "CW" (Certified-Withdrawn) to "C" (Confirmed) and removing all withdrawn cases ("W"), we noted that confirmed petitions comprised roughly 97% of the original 3,000,000 rows. To balance computational efficiency with model performance, and to ensure enough denial examples, we selected a 500,000-row subset. This subset comprised 80,000 denied cases (approximately all of them) and a random sample of 420,000 confirmed cases. We then shuffled and split this subset into 300,000 rows for training, 100,000 for validation, and 100,000 for testing.

```
# Sampling and Partitioning
set.seed(46524)
h1b_full <- read.csv("h1b_data.csv")
h1b_full$case_status[h1b_full$case_status == "CW"] <- "C"
h1b_full <- h1b_full[h1b_full$case_status != "W", ]

idx_C <- which(h1b_full$case_status == "C")
idx_D <- which(h1b_full$case_status == "D")
```

```r
sample_C <- sample(idx_C, 420000)
sample_D <- sample(idx_D, 80000)

sampled_idxs <- c(sample_C, sample_D)

h1b_subset <- h1b_full[sampled_idxs,]
h1b_subset <- h1b_subset[sample(nrow(h1b_subset)),]

train_df <- h1b_subset[1:300000,]
validation_df <- h1b_subset[300001:400000,]
test_df <- h1b_subset[400001:500000,]
```

```r
# Transformation Pipeline - Data Preprocessing
cols_to_remove <- c("emp_country", "decision_date", "wage_to","pw_level")
train_df <- train_df[, !(names(train_df) %in% cols_to_remove)]
validation_df <- validation_df[, !(names(validation_df) %in% cols_to_remove)]
test_df <- test_df[, !(names(test_df) %in% cols_to_remove)]

validation_df <- validation_df %>% filter(!is.na(case_status))
test_df <- test_df %>% filter(!is.na(case_status))

H1B_recipe <- recipe(case_status ~ ., data=train_df)%>%
  step_string2factor(all_nominal_predictors()) %>%
  step_mutate(case_submitted = as.Date(as.character(case_submitted), "%Y-%m-%d"))%>%
  step_mutate(case_year = factor(case_year),
              case_month = factor(lubridate::month(case_submitted)),
              case_wday = lubridate::wday(case_submitted, label = TRUE)) %>%
  step_rm(case_submitted) %>%
  step_other(all_nominal_predictors(),
             -all_of(c("emp_name","job_title","soc_name", "emp_willful_violator")),
             threshold = 0.01, other = "other")%>%
  step_mutate(emp_name = fct_relabel(emp_name, make.names),
              job_title = fct_relabel(job_title, make.names),
              soc_name = fct_relabel(soc_name, make.names)) %>%
  step_mutate(emp_name = fct_lump_n(emp_name, n = 50, other_level = "other"),
              job_title = fct_lump_n(job_title, n = 50, other_level = "other"),
              soc_name = fct_lump_n(soc_name, n = 50, other_level = "other")) %>%
  step_unknown(emp_h1b_dependent, emp_willful_violator,
               full_time_position, new_level = "unknown") %>%
  step_impute_median(all_numeric_predictors())%>%
  step_novel(all_nominal_predictors(), new_level = "novel") %>%
  step_impute_mode(all_nominal_predictors())%>%
  step_normalize(all_numeric_predictors())%>%
  step_corr(all_numeric_predictors(), threshold = 0.9)

H1B_recipe_prep <- prep(H1B_recipe, training = train_df, retain = TRUE)
```

```r
# "Baking" Subsets
train_processed <- bake(H1B_recipe_prep, new_data = NULL)
validation_processed <- bake(H1B_recipe_prep, new_data = validation_df)
test_processed <- bake(H1B_recipe_prep, new_data = test_df)
```

### 1 - Baseline Model

Our baseline model is a simple logistic regression on the predictors "*prevailing_wage*", "*full_time_position*", and "*case_year*". We select the task as classification for the output variable *case_status* (taking either "C" for confirmed and "D" for denied). This model is fitted on the training data and then the estimated coefficients for the predictors are shows with their p values and odd ratios. For the categorical variables, each level's coefficient is estimated separately.

```
# Train Logistic/Baseline Model
model_recipe <- recipe(case_status ~ prevailing_wage + full_time_position
                       + case_year , data = train_processed)
logistic <- logistic_reg(mode='classification') %>% set_engine('glm')
model_workflow <- workflow() %>% add_model(logistic) %>% add_recipe(model_recipe)
visa_lm <- fit(model_workflow, data = train_processed)
```

### 2 - Gradient Descent Model

For the gradient descent model part, we chose to implement a plain logistic-regression classifier using our own mini-batch gradient-descent loop as a transparent model. At each iteration, the model computes predicted probabilities via a sigmoid function and then adjusts its coefficients to reduce the average negative cross-entropy on small random batches. We experimented with different learning rates, batch sizes, and epoch counts-tracking train, validation, and test negative cross-entropy curves to guard against overfitting. Finally, we tuned the probability threshold on the validation set to optimise accuracy rather than defaulting to 0.5. This transparent implementation allows us to see exactly how each predictor shifts the odds of "C" versus "D."

```
# Design Matrices
X_train <- model.matrix(case_status ~ .,data = train_processed)
y_train <- if_else(train_processed$case_status == "C", 1, 0)
X_val <- model.matrix(case_status ~ ., data = validation_processed)
y_val <- if_else(validation_processed$case_status == "C", 1, 0)
X_test <- model.matrix(case_status ~ ., data = test_processed)
```

```
# Helper functions
sigmoid <- function(z) {1 / (1 + exp(-z))}
log_loss <- function(p, y) {-mean(y * log(p) + (1 - y) * log(1 - p))}
```

```
# Gradient Descent Algorithm
set.seed(46524)
gd_algorithm <- function(X_train, y_train,
                         X_val, y_val,
                         X_test, y_test,
                         lr = 0.05,
                         batch_size = 512,
                         n_epochs = 50) {
  n <- nrow(X_train)
  p <- ncol(X_train)
  beta <- numeric(p)
  train_loss <- numeric(n_epochs)
  val_loss <- numeric(n_epochs)
  test_loss <- numeric(n_epochs)
  for (epoch in seq_len(n_epochs)) {
    idx <- sample(n)
    for (i in seq(1, n, by = batch_size)) {
      rows <- idx[i:min(i + batch_size - 1, n)]
      xb <- X_train[rows, , drop = FALSE]
```

```
      yb <- y_train[rows]
      phat <- sigmoid(xb %*% beta)
      grad <- crossprod(xb, phat - yb) / length(rows)
      beta <- beta - lr * grad}
    train_loss[epoch] <- log_loss(sigmoid(X_train%*% beta), y_train)
    val_loss[epoch] <- log_loss(sigmoid(X_val%*% beta), y_val)
    test_loss[epoch] <- log_loss(sigmoid(X_test%*% beta), y_test)}
  list(coefs = beta, train_loss = train_loss, val_loss = val_loss, test_loss  = test_loss)
  }
```

**Mini-Batch Gradient Descent Algorithm:**

We begin by setting all coefficients to zero and then loop over a fixed number of epochs. At the start of each epoch, we randomly shuffle the training indices and break the data into mini-batches. For each mini-batch, we calculate the predicted probabilities from the current coefficient values, compute the gradient of the negative cross-entropy loss with respect to those coefficients, and subtract a scaled version of that gradient (using our learning rate) to update the coefficients. Once every batch in an epoch has been processed, we use the full training, validation, and test sets to compute and store the current loss values, which lets us track how the model is improving (or beginning to overfit) as training proceeds.

```
# Gradient Descent Training
set.seed(46524)
gd_out <- gd_algorithm(
  X_train, y_train, X_val, y_val,
  X_test, if_else(test_processed$case_status == "C", 1, 0),
  lr = 0.1, batch_size = 128, n_epochs = 40)
```

**Fine-Tuning Approach**

To fine-tune our model, we performed two grid searches-first over learning rates {0.1, 0.01, 0.001}, batch sizes {128, 256, 512}, and epochs {40, 100}, then refining to learning rates {0.1, 0.3}, batch sizes {64, 128}, and epochs {20, 40}. We selected the combination lr = 0.1, batch_size = 128, n_epochs = 40 because it yielded the lowest validation loss without overfitting. After training, we calculated predicted probabilities on the validation set and scanned thresholds from 0.10 to 0.90, identifying 0.47 as the value that maximised validation accuracy. Choosing 0.48 instead of the default 0.50 helps balance sensitivity and specificity, leading to more reliable positive and negative predictions.

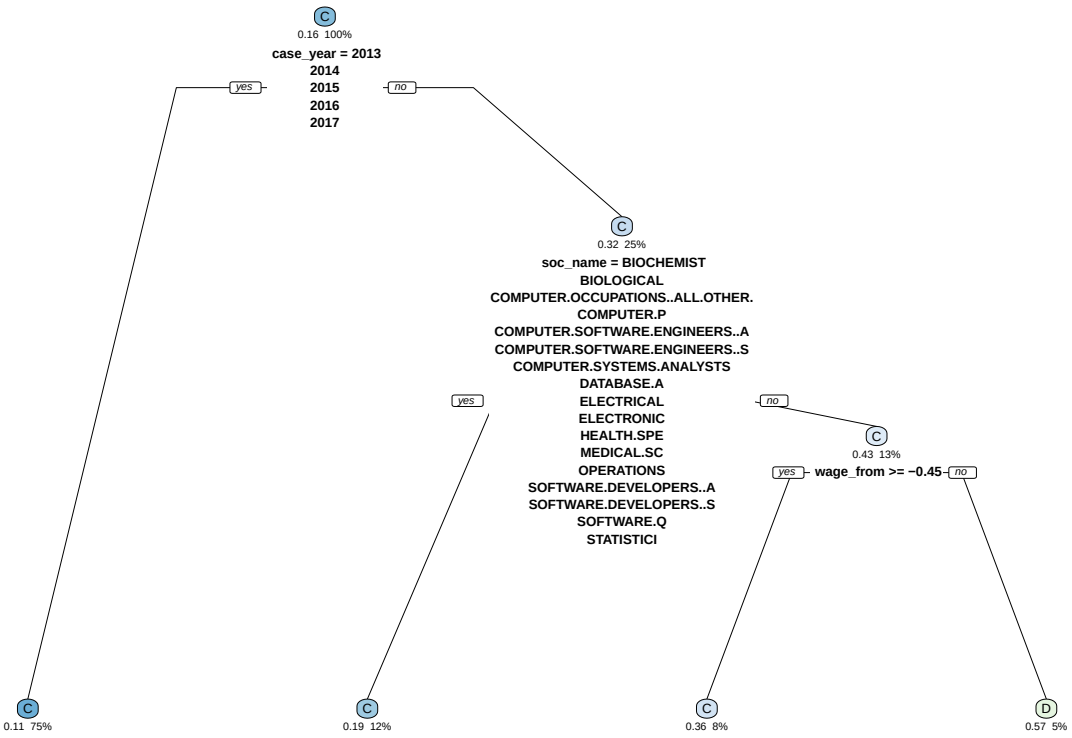
### 3 - Interpretable Model

We chose to fit a simple decision tree with depth 3 as our interpretable model since it is easy to understand for a non-technical audience. The visualisation of the tree gives easy-to-understand, real life demonstration. At each node, the model selects the variable and threshold that minimise impurity, and divides into two branches based on this if-else statements. Each subgroup's percentage in the population and their probability of getting denied for their visa is also visible in the tree graph.

```
# Train/Plot Decision Tree
set.seed(46524)
tree_workflow <- workflow() %>%
  add_formula(case_status ~ .) %>%
  add_model(decision_tree(mode = "classification", tree_depth = 3) %>% set_engine("rpart"))

final_tree <- fit(tree_workflow, data = train_processed)

rpart.plot(extract_fit_engine(final_tree), type = 2, fallen.leaves = TRUE, extra = 106,
           facsep = "\n", under = TRUE, yesno=2, faclen = 10,
           tweak = 1, compress = FALSE,branch=0.5)
```




**Interpretation:**

This decision tree has depth-3 and is fit on the training data, where the overall denial rate is 16%. It works by splitting the data based on simple conditions: whether the case belongs to given categories, or whether it exceeds given thresholds. Starting with the full data, it first asks if the year of the visa application is between 2013 and 2017. Seventy-five percent of the sample falls under this sub-tree, having an 11% chance of denial. In contrast, applicants from 2011 or 2012 have a 32% probability of denial. This branch is further divided based on whether the applicant falls under one of the mentioned Standard Occupational Classification jobs. If someone is classified under one of these jobs (e.g., BIOCHEMIST, COMPUTER.SOFTWARE.ENGINEERS…, STATISTICIAN), their denial rate drops to 19%, comprising 12% of cases. Otherwise, the rate increases to 43%, and it then checks whether their wage is over -0.45 standard deviations below the mean. If this condition is satisfied, the probability falls to 36% (8% of the population); if not, it increases to the maximum of 57%.

Compared with the baseline logistic-regression model, the depth-3 decision tree produces very similar diagnostics. However, the specificity rises sharply-from 3% to 16%, showing that the tree correctly flags about five times more denials while losing only a tiny portion of recall for approvals (97.6% vs. 99.6%). This cleaner trade-off happens because the tree can split applicants into narrower subgroups (by year, occupation, and wage) where a "Denied" label is genuinely more likely, instead of applying one fixed threshold to every case.

Logistic regression is linear in the log-odds: a one-unit increase in (standardised) prevailing wage multiplies the odds of approval by the same factor everywhere, regardless of year or occupation. This linearity, and the absence of interaction terms means the model can miss subtle patterns. The tree, by contrast, captures conditional effects: a higher than average wage may lower denial probability in one branch yet be irrelevant in another.

A new run on the same data may yield a slightly different tree, whereas the logistic coefficients are deterministic given the data. Nonetheless, the current tree offers a more balanced error profile and a clear set of "if-then" rules that reveal where linear assumptions break down. The baseline logistic regression, however, remains useful when you need a single, stable equation that is easy to interpret, when you value well-calibrated probability scores, or when you must explain effects globally rather than branch-by-branch.

### 4 - High-Dimensional Model

For our high-dimensional model, we create transformations to expand the predictor space. We convert "*case_year*" and "*case_month*" back to numeric, add quadratic terms for wages, encode the month as cyclical sine/cosine pairs, and create interaction terms such as wage × full-time, wage × year, and wage × location. We then generate natural spline bases (df = 3) on both "*prevailing_wage*" and "*case_year*" to capture smooth nonlinear trends. The same transformations are applied to training, validation, and test splits so that all three sets share identical columns for model fitting and evaluation.

```
# Preparing High-Dimensional Features
library(splines)
train_hd <- train_processed %>%
  mutate(
    case_year = as.integer(as.character(case_year)),
    case_month = as.integer(as.character(case_month)),
    prevailing_wage_sq = prevailing_wage^2,
    wage_from_sq = wage_from^2,
    month_sin = sin(2 * pi * case_month / 12),
    month_cos = cos(2 * pi * case_month / 12),
    wage_full = prevailing_wage * if_else(full_time_position == "Y", 1, 0),
    dep_viol = if_else(emp_h1b_dependent == "Y", 1, 0) *
      if_else(emp_willful_violator == "Y", 1, 0),
    wage_year  = prevailing_wage * case_year,
    wage_lat = prevailing_wage * lat,
    wage_lng = prevailing_wage * lng,
    wage_s1 = ns(prevailing_wage, df = 3)[,1],
    wage_s2 = ns(prevailing_wage, df = 3)[,2],
    wage_s3 = ns(prevailing_wage, df = 3)[,3],
    year_s1 = ns(case_year,df = 3)[,1],
    year_s2 = ns(case_year,df = 3)[,2],
    year_s3 = ns(case_year, df = 3)[,3])
```

Same is applied to validation and test datasets. (Check Appendix for the code)

#### Fine-Tuning Approach

We conducted a grid search over elastic-net parameters on the validation split ($\alpha \in \{0.5, 1.0\}$, $\log(\lambda) \in [-6, -3]$) and found that setting $\alpha = 1$ (pure Lasso) and $\lambda = 10^{-6}$ maximised ROC AUC $\approx 0.78$ on validation. Because these values gave the best trade-off between sparsity and predictive power, we hard-coded a Lasso logistic regression with $\alpha = 1$ and $\lambda = 10^{-6}$, due to extremely long running times. This single final specification ensures that only the most relevant features survive regularisation.

```
# Lasso Workflow
lasso_spec <- logistic_reg(
  penalty = 1e-6,
  mixture = 1) %>%
  set_engine("glmnet") %>%
  set_mode("classification")

lasso_wf <- workflow() %>%
  add_model(lasso_spec) %>%
  add_formula(case_status ~ .)
```

With the chosen hyperparameters in place, we fit the final Lasso on the training split only (300 K observations). This is then applied to the hold-out test set to obtain predicted probabilities and class labels (threshold = 0.5). Finally, we compute standard diagnostics.

### 5 - Prediction Competition Model

For our prediction competition model, we first define two custom functions to encapsulate functionality that we will use frequently. One function identifies the optimal threshold for converting predicted probabilities into class labels, based on the validation set. The other function computes and displays our standardised set of performance metrics for a model, given its test data.

```
# Helper functions
threshold_function <- function(model, data_val) {
  if (identical(class(model), c("glm", "lm"))) {
    tp = "response"} else {tp = "prob"}
  val_probs <- predict(model, data_val, type = tp) %>%
    bind_cols(data_val %>% select(case_status))
  thresholds <- seq(0.10, 0.90, by = 0.01)
  acc_by_thr <- tibble(thresh = thresholds) %>%
    mutate(
      accuracy = map_dbl(thresh, ~ mean(
        (val_probs$.pred_C > .x) == (val_probs$case_status == "C"))))
  best_threshold <- acc_by_thr %>%
    slice_max(accuracy, n = 1)
  print(best_threshold)
  return(best_threshold)
}

diagnostics_function <- function(model, test_data, validation_data) {
  if (identical(class(model), c("glm", "lm"))) {
    tp = "response"
  } else {
    tp = "prob"
  }
  test_probs <- predict(model, test_data, type = tp) %>%
    bind_cols(test_data %>% select(case_status))
  best_threshold <- threshold_function(model, validation_data)
  test_pred <- test_probs %>%
    mutate(pred = factor(if_else(.pred_C > best_threshold$thresh, "C", "D"),
                         levels = c("C","D")))
  diagnostics <- tibble(
    Metric = c("Accuracy","Sensitivity (Recall)", "Specicifity",
               "Precision", "F1 Score", "ROC AUC"),
    Value = c(accuracy_vec(test_pred$case_status, test_pred$pred),
              sensitivity(test_pred, truth = case_status, estimate = pred)$.estimate,
              specificity(test_pred, truth = case_status, estimate = pred)$.estimate,
              precision(test_pred, truth = case_status, estimate = pred)$.estimate,
              f_meas(test_pred, truth = case_status, estimate = pred)$.estimate,
              roc_auc_vec(
                truth = test_pred$case_status,
                estimate = as.numeric(test_pred$.pred_C))))
  print(diagnostics)
}
```

#### Model Architecture

This model is constructed by stacking three base learners-XGBoost, Random Forest, and GLM. In this approach, a meta-model is trained on the predicted probabilities produced by the base models. Our chosen meta-model is also an XGBoost. To implement stacking, we partitioned the training data into two subsets: 70% for fitting the base models and 30% for fitting the meta-model. The code below fits each of the three

base models using their respective tuned hyper-parameters.

Below is the code for out XGBoost model.

```
#XGB Model
train_indices <- sample(seq_len(nrow(train_processed)), size = 0.7 * nrow(train_processed))
train_processed_A <- train_processed[train_indices, ]
train_processed_B  <- train_processed[-train_indices, ]

library(xgboost)

xgb_spec <- boost_tree(
  trees = 500,
  tree_depth = tune(),
  learn_rate = tune(),
  loss_reduction = tune()
) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

xgb_wf <- workflow() %>%
  add_model(xgb_spec) %>%
  add_formula(case_status ~ .)

train_sample <- train_processed_A %>% sample_frac(1)
folds <- vfold_cv(train_sample, v = 3)

xgb_grid <- tibble(
  tree_depth = c(10, 9),
  learn_rate = c(0.01, 0.15),
  loss_reduction = c(10, 3))
xgb_grid <- expand_grid(
  tree_depth = c(10, 15),
  learn_rate = c(0.01, 0.001),
  loss_reduction = c(10, 20))

best_params <- tibble::tibble(tree_depth = 10, learn_rate = 0.01, loss_reduction = 10)
```

Below is the code for the Random Forest Model:

```
# Random Forest
library(ranger)
rf_spec <- rand_forest(
  trees = 1000,
  mtry = tune(),
  min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")

rf_wf <- workflow() %>%
  add_model(rf_spec) %>%
  add_formula(case_status ~ .)

train_sample <- train_processed_A %>% sample_frac(1)
folds <- vfold_cv(train_sample, v = 3)
```

```
rf_grid <- expand_grid(
  mtry = c(5),
  min_n = c(10))

best_params <- tibble::tibble(mtry = 5, min_n = 10)
```

And finally the code for the GLM:

```
# GLM
glm_spec <- logistic_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet") %>%
  set_mode("classification")

glm_wf <- workflow() %>%
  add_model(glm_spec) %>%
  add_formula(case_status ~ .)

train_sample <- train_processed_A %>% sample_frac(1)

folds <- vfold_cv(train_sample, v = 3)

glm_grid <- expand_grid(
  penalty = 10^seq(-8, -4, length.out = 5),
  mixture = c(0, 1))

best_params <- tibble::tibble(penalty = 1e-4, mixture = 1)
```

With the three base models in place, we now stack them by training an XGBoost meta-model on their predicted probabilities. Up to this point, each base model's performance has been evaluated using diagnostics_function(). In the code below, we apply the same evaluation procedure to the stacked meta-model.

```
# Model stacking with XGBoost
stack_train <- tibble(
  xgb = xgb_probs$.pred_C,
  rf = rf_probs$.pred_C,
  glm = glm_probs$.pred_C,
  case_status = train_processed_B$case_status)

stack_validation <- tibble(
  xgb = xgb_validation$.pred_C,
  rf = rf_validation$.pred_C,
  glm = glm_validation$.pred_C,
  case_status = validation_processed$case_status)

stack_test <- tibble(
  xgb = xgb_test$.pred_C,
  rf = rf_test$.pred_C,
  glm = glm_test$.pred_C,
  case_status = test_processed$case_status)

meta_xgb_spec <- boost_tree(
  trees = 500,
  tree_depth = tune(),
  learn_rate = tune(),
  loss_reduction = tune()
```

```
  ) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

meta_xgb_wf <- workflow() %>%
  add_model(meta_xgb_spec) %>%
  add_formula(case_status ~ .)

train_sample <- stack_train %>% sample_frac(1)
folds <- vfold_cv(train_sample, v = 3)

meta_xgb_grid <- expand_grid(
  tree_depth = c(2, 3),
  learn_rate = c(0.1, 0.05),
  loss_reduction = c(10))

best_params <- tibble::tibble(tree_depth = 2, learn_rate = 0.1, loss_reduction = 10)
```

**Model Comparison and Ensemble Strategy**

Several individual learners-XGBoost, Random Forest, GLM, and a neural network-were fitted and tuned using a 10% random sample of the training data to speed up hyper-parameter searches. Each model underwent grid search on that sample, and their validation performances were compared. The neural network proved difficult to train and was ultimately discarded. XGBoost emerged as the strongest single model.

For each candidate model, a custom function scanned a range of probability thresholds on the validation set and selected the threshold that maximized accuracy. This ensured a fair comparison among models before moving on to ensemble methods. A simple majority-vote ensemble was then attempted across the three tuned base models: for each test observation, the class predicted by at least two of the three models was chosen. However, this voting approach failed to improve accuracy over XGBoost alone and actually reduced specificity. Because each base learner already favored predicting "C" about 85% of the time (due to the pronounced class imbalance), majority voting simply amplified that bias and further degraded specificity.

After ruling out voting, stacking was pursued. First, a GLM meta-model was trained on the base models' predicted probabilities, but its performance did not exceed that of the XGBoost base learner. The GLM was then replaced with an XGBoost meta-model, which was tuned via grid search. This stacked XGBoost outperformed every individual base model across most metrics (sensitivity was already above 97% for all base learners, leaving no room for improvement on that metric).

Once the meta-model and all base models were tuned, they were retrained on the full training dataset (instead of the 10% sample) and evaluated on the test set. The resulting diagnostics were:

These results represent the culmination of successive refinements, confirming that the stacked XGBoost model delivers the best overall balance of accuracy and discrimination.

**Table of Diagnostics for Predictions on Test Data**

| Metric | Baseline | Gradient Descent | Interpretable | High Dimensional | Prediction Competition |
|---|---|---|---|---|---|
| Accuracy | 0.84110 | 0.85109 | 0.84407 | 0.85200 | 0.86758 |
| Sensitivity (Recall) | 0.99623 | 0.98147 | 0.97558 | 0.98000 | 0.97053 |
| Specificity | 0.03141 | 0.16366 | 0.15766 | 0.18000 | 0.32474 |
| Precision | 0.84297 | 0.86087 | 0.85805 | 0.86200 | 0.88343 |
| F1 Score | 0.91322 | 0.91722 | 0.91305 | 0.91700 | 0.92493 |
| ROC AUC | 0.68592 | 0.77602 | 0.66482 | 0.78200 | 0.83280 |

## Results

**Accuracy**
Baseline logistic regression and a shallow decision tree achieve similar accuracy (84.11 % and 84.41 %). Mini-batch gradient-descent logistic regression (85.11%) and a high-dimensional Lasso (85.20%) improves on them moderately. The stacked ensemble of XGBoost, Random Forest, and GLM achieves the highest accuracy at 86.76% by combining complementary models.

**Sensitivity (Recall)**
Because the baseline logistic regression classifies the vast majority of cases as "Confirmed," it attains an unsurprisingly high recall of 99.62%. The remaining single-model approaches exhibit similarly strong recall, between 97.5% and 98.5%. The stacked ensemble records the lowest recall (97.05%), reflecting its trade-off of a small reduction in sensitivity for a substantial gain in specificity.

**Specificity**
The baseline logistic regression detected denials poorly, with a specificity of only 3.14%. Improving this measure therefore became a key objective. The decision tree, gradient-descent logistic regression, and Lasso models each raise specificity to 15.76%, 16.37%, and 18.00%, respectively. The stacked ensemble pushes specificity to 32.47%, more than doubling the best single-model result.

**Precision**
Logistic regression has a precision of 84.30 percent. The decision tree filters out more obvious denial cases via its if-then splits, yielding higher precision (85.81 percent) than baseline. The gradient descent and the high-dimensional Lasso model achieves slightly better precision with 86.09 percent and 86.20 percent respectively. The prediction competition model attains the highest precision with 88.34 percent.

**F1 Score**
The first four models (logistic regression, decision tree, gradient-descent variant, and Lasso) all achieve very similar F1 scores (around 91.3-91.7 percent), while the stacked ensemble outperforms them with an F1 of 92.49 percent.

**ROC AUC**
Logistic regression achieves a 68.59 percent AUC, and the decision tree is slightly lower at 66.48 percent due to limited generalisation. Mini-batch gradient descent improves ranking to 77.60 percent, and the high-dimensional Lasso model raises it to 78.20 percent. The stacked ensemble attains the highest AUC of 83.28 percent.

**Why Specificity Is Critical**
In visa approval tasks, a false approval can have severe consequences, whether a security risk or a legal violation. Because denials are rare (about 17 percent of applications), a model that always predicts "approve" would achieve 83 percent accuracy without learning anything useful. Improving specificity forces the model to learn denial patterns rather than default to approval. Our sampling strategy addressed this challenge by training on 500,000 cases that include all 80,000 denials, ensuring the model saw enough negative examples. However, the simple logistic regression still had only 3.14 percent specificity. By contrast, the stacked ensemble reached 32.47 percent, roughly tripling baseline performance and doubling the specificity of other models.

Ultimately, higher specificity means fewer false approvals. In an environment where approving someone who should be denied is a real risk, sacrificing a bit of recall to boost denial detection is essential. Specificity ensures the model does not just learn that most applicants are approved, but it forces the model to recognise subtle denial indicators such as lower employer credentials.

**Interpretability vs. Black-Box Models in Visa Approvals**
Automated visa decisions must balance performance with transparency. Our stacked ensemble, which layers XGBoost, Random Forest, and GLM under a final XGBoost meta-learner, delivers top metrics, accuracy, AUC, and improved specificity, but it behaves as a black box. It is nearly impossible to explain an individual decision beyond saying "the data suggested approval or denial." If an applicant or regulator asks "why was I denied?", the ensemble cannot point to a single clear rule. By contrast, a shallow decision tree provides complete transparency. Anyone can trace exactly why an application was approved or denied. That makes audits and

legal compliance straightforward. However, the tree's simplicity sacrifices predictive power: it cannot capture the complex interactions that the ensemble does. When false approvals carry serious implication, security, legal, or ethical, a transparent approach can be more valuable than marginal accuracy gains.

### Significance Levels of Predictors

To further investigate how significance each predictor is while predicting the output class, we fitted a logistic regression model with all the predictors.

```
# Logistic model for all predictors
model_recipe <- recipe(case_status ~ . , data = train_processed)
logistic <- logistic_reg(mode='classification') %>% set_engine('glm')
model_workflow <- workflow() %>% add_model(logistic) %>% add_recipe(model_recipe)
visa_lm <- fit(model_workflow, data = train_processed)
```

We then checked the top 20 most significant predictors.

## Table for Top 20 Significant Predictors

| predictor | estimate | p.value | odds_ratio |
|---|---|---|---|
| case_year2013 | -0.9989276 | 0.000000e+00 | 3.682742e-01 |
| case_year2014 | -1.1793906 | 0.000000e+00 | 3.074660e-01 |
| pw_unitY | -3.0161168 | 0.000000e+00 | 4.899109e-02 |
| wage_unitY | 2.3959558 | 4.727962e-241 | 1.097869e+01 |
| wage_from | -0.6329940 | 6.047936e-182 | 5.309996e-01 |
| prevailing_wage | 20.2190124 | 9.872242e-152 | 6.039563e+08 |
| case_year2012 | -0.4551093 | 4.573174e-123 | 6.343786e-01 |
| wage_unitother | 2.6644402 | 1.042714e-114 | 1.435991e+01 |
| emp_h1b_dependentY | -0.4380906 | 1.648677e-63 | 6.452673e-01 |
| job_titlePROGRAMMER.ANALYST | -1.0165544 | 1.767038e-50 | 3.618395e-01 |
| job_titleASSISTANT.PROFESSOR | -1.1625947 | 5.209051e-42 | 3.126738e-01 |
| pw_unitother | -1.7015601 | 3.845701e-39 | 1.823987e-01 |
| job_titleSYSTEMS.ANALYST | -0.9021486 | 2.007666e-28 | 4.056970e-01 |
| job_titleRESEARCH.ASSOCIATE | -1.1031467 | 5.710785e-27 | 3.318253e-01 |
| case_month5 | -0.3252827 | 6.211469e-27 | 7.223231e-01 |
| emp_nameother | 2.1918466 | 4.427059e-26 | 8.951728e+00 |
| emp_stateNJ | -0.3738991 | 5.835603e-25 | 6.880463e-01 |
| job_titleBUSINESS.ANALYST | -0.8104042 | 1.681373e-22 | 4.446783e-01 |
| case_wday.Q | 0.3259441 | 1.778813e-22 | 1.385338e+00 |
| emp_nameERNST_AND_YOUNG_LLP | 2.1250780 | 1.714642e-21 | 8.373550e+00 |

**Key Findings:**

- Applications submitted in 2013 and 2014 had substantially lower odds of approval compared to 2011 (odds ratios around 0.3 to 0.4).
- Higher values of prevailing wage had a very large positive effect on the likelihood of approval, although this extremely high odds ratio appears due to numeric scaling of the data.
- Employer characteristics were also influential. Specifically, applications with employer name "Ernst and Young LLP" were eight times more likely to get accepted compared to the baseline employer. As this is a well-known, large company, it intuitively makes sense that they could be more reliable.

- Applications from H-1B-dependent employers had moderately reduced odds (odds ratio approximately 0.65).

- Positions such as Research Associate, Systems Analyst, or Business Analyst had a lower likelihood of approval. However, this result may be biased due to the sample sizes of these roles and because these odds are compared to baseline values, which should be further investigated.

## Conclusion

Our goal was to leverage machine learning to predict H-1B visa outcomes to improve transparency and consistency in the application process. We used a large, labeled dataset of H-1B petitions (2011-2018) to apply five different machine learning approaches: a baseline logistic regression model for simplicity, a mini-batch gradient-descent logistic regression for optimisation, a decision tree of depth 3 for ease of interpretability, a high-dimensional Lasso model with polynomials, interactions, and splines to capture nonlinear effects, and a model targeting maximum predictive accuracy.

Our analysis does have some limitations. The dataset reflects only those applications that reached a final "approved" or "denied" status, so it omits petitions that were withdrawn or remain pending. It also relies on the information fields reported at filing-unobserved factors (such as detailed employer compliance history or nuanced legal arguments) are not captured. Moreover, because our sample was constructed by subsampling (420 K approvals vs. 80 K denials) and then randomized, the absolute approval rates in our training set differ from the true population proportions. Although we explored ways to adjust model outputs for that imbalance, additional calibration or reweighting would be required before deployment to ensure predictions reflect real-world proportions.

Looking ahead, these models could be integrated with real-time regulatory data (such as Department of Labor audits, evolving wage requirements, or changes in immigration policy) to support government entities in flagging high-risk applications or allocating resources more efficiently. Because none of our features include race, nationality, or other protected attributes, our predictions remain relatively unbiased. We believe this is vital to safeguard applicants from discrimination when deploying a black-box algorithm to decide visa outcomes. By integrating accurate predictions with legal and policy requirements, future work can develop a decision-support tool that directs adjudicators' attention to the most ambiguous or potentially non-compliant cases.

## References


1. H-1B Non-Immigrant Labour Visa. (2023, February 11). Kaggle. https://www.kaggle.com/datasets/thedevastator/h-1b-non-immigrant-labour-visa
2. H-1B Workflow | Florida Tech. (n.d.). https://www.fit.edu/international-student-and-scholar-services/for-departments/h-1b-visa-program/h-1b-workflow/
3. H-1B Electronic Registration Process | USCIS. (2025, May 16). USCIS. https://www.uscis.gov/working-in-the-united-states/temporary-workers/h-1b-specialty-occupations/h-1b-electronic-registration-process


## Appendix

The following appendix provides the additional R code chunks that were executed during our analysis but omitted from the main report to ensure full reproducibility.

```
glm_fit <- extract_fit_parsnip(visa_lm)$fit
coefficients <- tidy(glm_fit, conf_int=TRUE) %>%
  mutate(odds_ratio = exp(estimate)) %>%
  select(term, estimate, p.value, odds_ratio)
coefficients
```

```r
# Fine-Tuning High-Dimensional Model
enet_spec <- logistic_reg(
  penalty = tune(),
  mixture = tune()
) %>%
  set_engine("glmnet") %>%
  set_mode("classification")

enet_wf <- workflow() %>%
   add_model(enet_spec) %>%
   add_formula(case_status ~ .)

enet_grid <- grid_regular(
  penalty(range = c(-6, -3)),
   mixture(range = c(0.5, 1.0)),
   levels = c(penalty = 3, mixture = 3))

valid_rset <- validation_split(validation_hd, prop = 1.0, strata = case_status)

enet_res <- tune_grid(
enet_wf,resamples = valid_rset,
grid = enet_grid,
metrics = metric_set(roc_auc),
control = control_grid(verbose = TRUE))
```

```r
# XGB Fine Tuning
 xgb_tune <- tune_grid(
   xgb_wf,
   resamples = folds,
   grid = xgb_grid,
   metrics = metric_set(accuracy, roc_auc),
   control = control_grid(
     verbose = TRUE,
     save_pred = TRUE))

 best_params <- select_best(xgb_tune, metric = "accuracy")
 show_best(xgb_tune, metric = "accuracy", n = Inf)
```

```r
# Random Forest Fine-Tuning
rf_tune <- tune_grid(
  rf_wf,
  resamples = folds,
  grid = rf_grid,
  metrics = metric_set(accuracy, roc_auc),
  control = control_grid(
    verbose = TRUE,
    save_pred = TRUE))

best_params <- select_best(rf_tune, metric = "accuracy")
show_best(rf_tune, metric = "accuracy", n = Inf)
```

```r
# GLM Fine-Tuning
glm_tune <- tune_grid(
  glm_wf,
```

```
  resamples = folds,
  grid = glm_grid,
  metrics = metric_set(accuracy, roc_auc),
  control = control_grid(verbose = TRUE, save_pred = TRUE))

best_params <- select_best(glm_tune, metric = "accuracy")
show_best(glm_tune, metric = "accuracy", n = Inf)
```

```
#Meta Fine Tuning
meta_xgb_tune <- tune_grid(
   meta_xgb_wf,
   resamples = folds,
   grid = meta_xgb_grid,
   metrics = metric_set(accuracy, roc_auc),
   control = control_grid(
     verbose = TRUE,
     save_pred = TRUE))

best_params <- select_best(meta_xgb_tune, metric = "accuracy")
show_best(meta_xgb_tune, metric = "accuracy", n = Inf)
```

```
# Model Diagnostics
library(tidymodels)
p_test <- as.numeric(sigmoid(X_test %*% gd_out$coefs))

test_probabilities <- tibble(.pred_C = p_test) %>%
  bind_cols(test_processed %>% select(case_status))

test_predictions <- test_probabilities %>%
  mutate(pred = factor(if_else(.pred_C > best_thr, "C", "D"), levels = c("C", "D")))

diagnostics <- tibble(
  Metric = c("Accuracy", "Sensitivity (Recall)", "Specificity",
             "Precision","F1 Score","ROC AUC"),
  Value  = c(
    accuracy_vec(test_predictions$case_status, test_predictions$pred ),
    sensitivity(test_predictions, truth = case_status, estimate = pred )$.estimate,
    specificity(test_predictions, truth = case_status, estimate = pred )$.estimate,
    precision(test_predictions, truth = case_status, estimate = pred )$.estimate,
    f_meas(test_predictions, truth = case_status, estimate = pred )$.estimate,
    roc_auc_vec(test_predictions$case_status, test_predictions$.pred_C )))

diagnostics
```

```
# Model Diagnostics
test_probs <- predict(final_tree, test_processed, type = "prob") %>%
  bind_cols(test_processed %>% select(case_status))

test_pred <- test_probs %>%
  mutate(pred = factor(if_else(.pred_C > 0.5, "C", "D"), levels = c("C","D")))

diag <- tibble(
  Metric = c("Accuracy","Sensitivity (Recall)","Specificity","Precision","F1 Score","ROC AUC"),
  Value = c(accuracy_vec(test_pred$case_status, test_pred$pred),
```

```
    sensitivity_vec(test_pred$case_status, test_pred$pred),
    specificity_vec(test_pred$case_status, test_pred$pred),
    precision_vec(test_pred$case_status, test_pred$pred),
    f_meas_vec(test_pred$case_status, test_pred$pred),
    roc_auc_vec(test_probs$case_status, test_probs$.pred_C)))
diag
```

```
# Model diagnostics
set.seed(46524)
final_lasso_fit <- fit(lasso_wf, data = train_hd)

test_probabilities <- predict(final_lasso_fit, test_hd, type = "prob") %>%
  bind_cols(test_processed %>% select(case_status))

test_predictions <- test_probabilities %>%
  mutate(pred = factor(if_else(.pred_C > 0.5, "C", "D"),levels = c("C","D")))

diagnostics <- tibble(
  Metric = c("Accuracy","Sensitivity (Recall)", "Specicifity",
             "Precision", "F1 Score", "ROC AUC"),
  Value = c(accuracy_vec(test_predictions$case_status, test_predictions$pred),
            sensitivity(test_predictions, truth = case_status, estimate = pred)$.estimate,
            specificity(test_predictions, truth = case_status, estimate = pred)$.estimate,
            precision(test_predictions, truth = case_status, estimate = pred)$.estimate,
            f_meas(test_predictions, truth = case_status, estimate = pred)$.estimate,
            roc_auc_vec(test_predictions$case_status, test_predictions$.pred_C)))
diagnostics
```

```
# XGB workflow and diagnostics
final_xgb_wf <- finalize_workflow(xgb_wf, best_params)
final_xgb_fit <- fit(final_xgb_wf, data = train_sample)

xgb_probs <- predict(final_xgb_fit, train_processed_B, type = "prob")
xgb_validation <- predict(final_xgb_fit, validation_processed, type = "prob")
xgb_test <- predict(final_xgb_fit, test_processed, type = "prob")

diagnostics_function(final_xgb_fit, test_processed, validation_processed)
```

```
# Random Forest workflow and model diagnostics
final_rf_wf <- finalize_workflow(rf_wf, best_params)
final_rf_fit <- fit(final_rf_wf, data = train_sample)

rf_probs <- predict(final_rf_fit, train_processed_B, type = "prob")
rf_validation <- predict(final_rf_fit, validation_processed, type = "prob")
rf_test <- predict(final_rf_fit, test_processed, type = "prob")

diagnostics_function(final_rf_fit, test_processed, validation_processed)
```

```
#GLM workflow and diagnostics
final_glm_wf <- finalize_workflow(glm_wf, best_params)
final_glm_fit <- fit(final_glm_wf, data = train_sample)

glm_probs <- predict(final_glm_fit, train_processed_B, type = "prob")
glm_validation <- predict(final_glm_fit, validation_processed, type = "prob")
glm_test <- predict(final_glm_fit, test_processed, type = "prob")
```

```r
diagnostics_function(final_glm_fit, test_processed, validation_processed)
```

```r
# Meta Model workflow and diagnostics
final_meta_xgb_wf <- finalize_workflow(meta_xgb_wf, best_params)
meta_xgb_fit <- fit(final_meta_xgb_wf, data = train_sample)
diagnostics_function(meta_xgb_fit, stack_test, stack_validation)
```

```r
# Top 20 significant predictors

library(dplyr)
glm_fit <- extract_fit_parsnip(visa_lm)$fit
coefficients <- tidy(glm_fit, conf_int=TRUE) %>%
  mutate(odds_ratio = exp(estimate)) %>%
  select(term, estimate, p.value, odds_ratio) %>%
  arrange(p.value)
top20 <- coefficients %>% dplyr::slice(1:20)
top20
```

```r
validation_hd <- validation_processed %>%
  mutate(
    case_year = as.integer(as.character(case_year)),
    case_month = as.integer(as.character(case_month)),
    prevailing_wage_sq = prevailing_wage^2,
    wage_from_sq = wage_from^2,
    month_sin = sin(2 * pi * case_month / 12),
    month_cos = cos(2 * pi * case_month / 12),
    wage_full = prevailing_wage * if_else(full_time_position == "Y", 1, 0),
    dep_viol  = if_else(emp_h1b_dependent == "Y", 1, 0)
    * if_else(emp_willful_violator == "Y", 1, 0),
    wage_year = prevailing_wage * case_year,
    wage_lat  = prevailing_wage * lat,
    wage_lng  = prevailing_wage * lng,
    wage_s1 = ns(prevailing_wage, df = 3)[,1],
    wage_s2 = ns(prevailing_wage, df = 3)[,2],
    wage_s3 = ns(prevailing_wage, df = 3)[,3],
    year_s1 = ns(case_year, df = 3)[,1],
    year_s2 = ns(case_year, df = 3)[,2],
    year_s3 = ns(case_year, df = 3)[,3])

test_hd <- test_processed %>%
  mutate(
    case_year = as.integer(as.character(case_year)),
    case_month = as.integer(as.character(case_month)),
    prevailing_wage_sq = prevailing_wage^2,
    wage_from_sq = wage_from^2,
    month_sin = sin(2 * pi * case_month / 12),
    month_cos = cos(2 * pi * case_month / 12),
    wage_full = prevailing_wage * if_else(full_time_position == "Y", 1, 0),
    dep_viol = if_else(emp_h1b_dependent == "Y", 1, 0) * if_else(emp_willful_violator == "Y", 1, 0),
    wage_year = prevailing_wage * case_year,
    wage_lat = prevailing_wage * lat,
    wage_lng = prevailing_wage * lng,
    wage_s1 = ns(prevailing_wage, df = 3)[,1],
    wage_s2 = ns(prevailing_wage, df = 3)[,2],
```

```
    wage_s3 = ns(prevailing_wage, df = 3)[,3],
    year_s1 = ns(case_year, df = 3)[,1],
    year_s2 = ns(case_year, df = 3)[,2],
    year_s3 = ns(case_year, df = 3)[,3])
```