Election Winner Classification Report

Class : Statistics 101C

Group : Team 14

Members : Claire Nabours (206014740), Jackie Sung (806062422),
Naikang Wang (206791396), Yankai Wen (606792233）

Introduction

Understanding county-level voting patterns is central to contemporary electoral analysis. In this project, we develop a machine-learning classifier to predict the winning candidate for each U.S. county using demographic and educational indicators from the U.S. Census Bureau. It is widely understood that voting behaviors often correlate with various socio-demographic factors. Accordingly, we hypothesize that variables such as education level, racial composition, and population density will be crucial predictive factors in our model. We aim for accurate, interpretable predictions that reflect established patterns in American elections.

Exploratory Data Analysis

In this section we will examine the datasets structure and quality to aid our goal of correctly classifying the winner between Trump and Biden. We will review the outcome distribution, distribution of predictors, simple relationships, and interactions between variables in our train_class.csv. These findings will motivate our preprocessing, transformation, and feature encoding choices and will later be used in the preprocessing section and help us with specifications when creating our models.

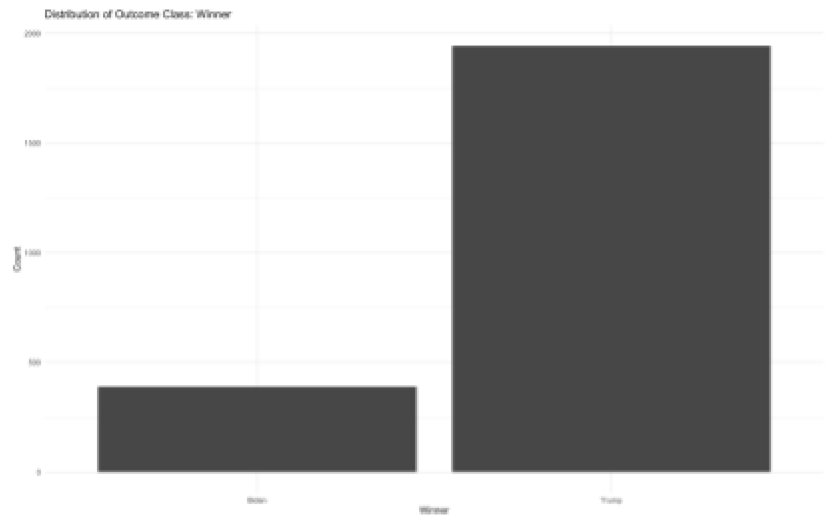


Figure 1: Bar chart of proportion of Trump vs. Biden wins

We begin by examining the outcome class proportions in Figure 1. The bar chart highlights a strong imbalance, with Biden representing 16.7% of observations and Trump with 83.3%. Careful consideration during preprocessing will be taken as models may favor the majority class.

We will emphasize balanced metrics such as ROC AUC rather than accuracy alone to ensure sensitivity to Biden wins while maintaining overall performance.

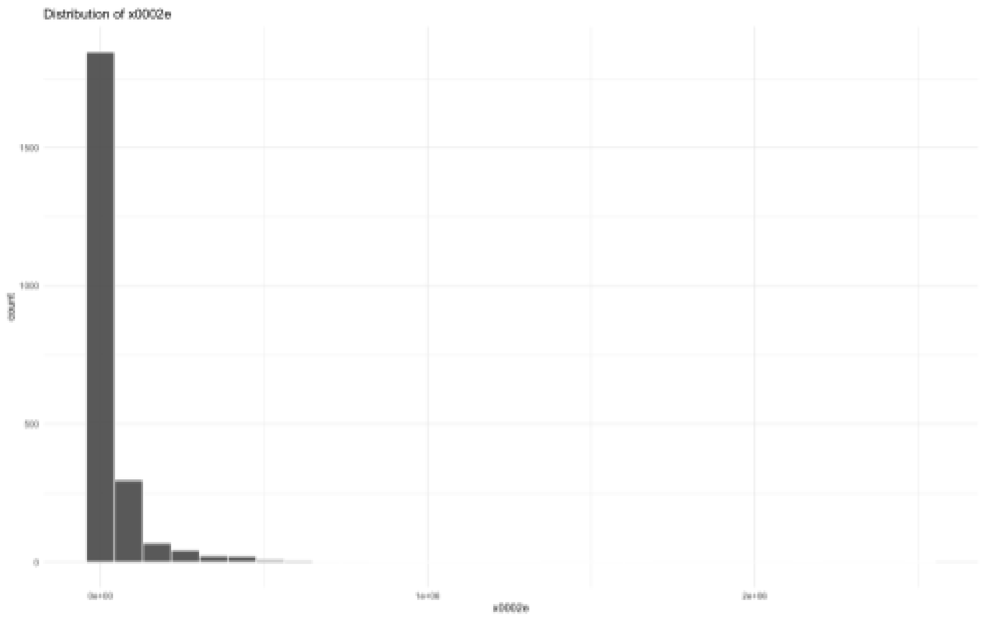


Figure 2: Distribution of estimate of total male population

In Figure 2, the male population count is highly right-skewed, a pattern shared by many other count variables.To address both scale and skew, we impute missing values, apply Yeo–Johnson or log transformations to counts, standardize with normalization, and favor proportions such as male share over raw counts for modeling.

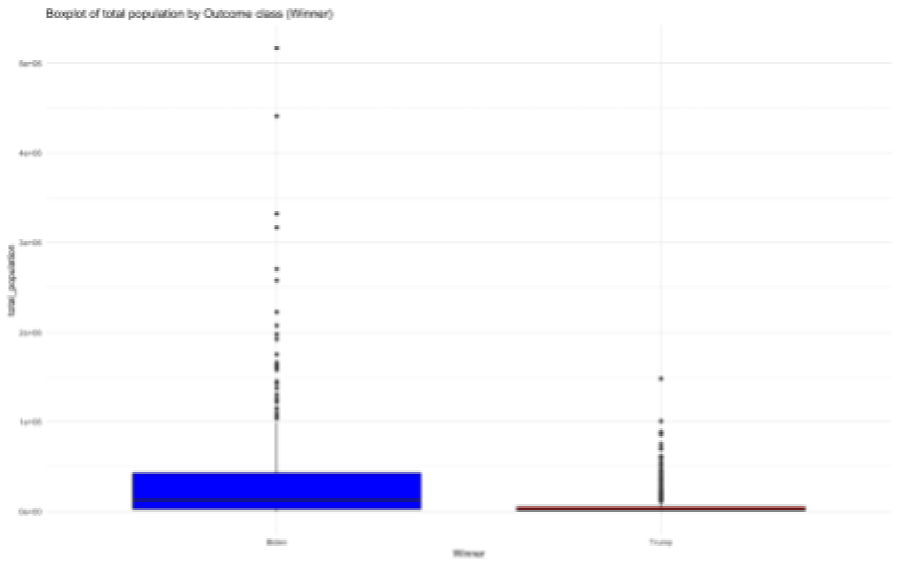


Figure 3: total population by winner

The box plot in figure 3 shows that counties labeled Biden have much larger populations, with higher medians, while Trump counties are mostly small. This implies Biden tends to win in more populous or urban areas and Trump tends to win in less populous areas. To compare other factors fairly, we will use proportions instead of raw counts to better normalize and account for county size and reveal the relative differences.

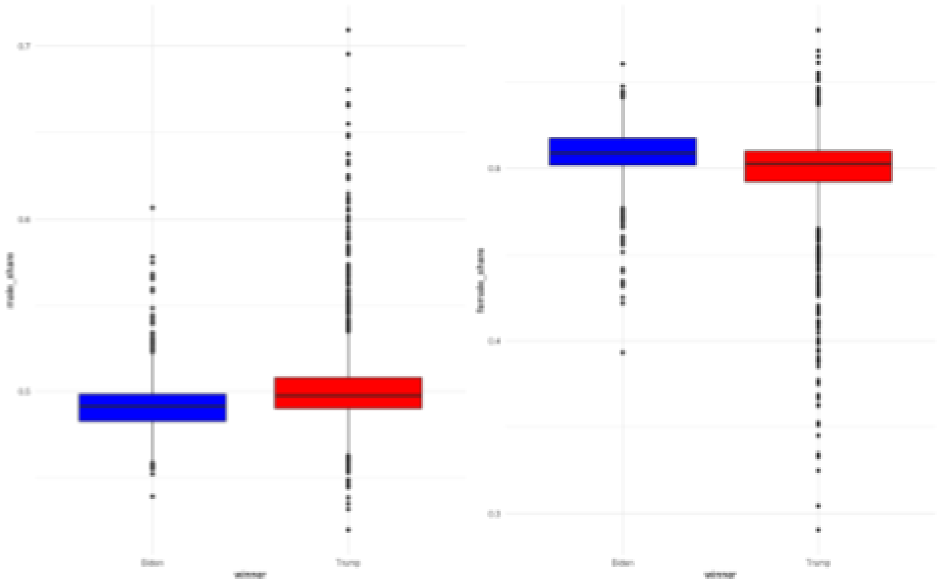


Figure 4: Proportion of male and female voters by winner

For example, in Figure 4, we can see differences in gender composition through boxplots of the proportion of female and male voters over total population. Counties that went for Trump tend to have a slightly higher male share and lower female shares. On the other hand, counties Biden won skew a bit more female. However there's still a lot of overlap, so gender by itself doesn't determine the outcome however it may influence results when considered alongside other predictors.

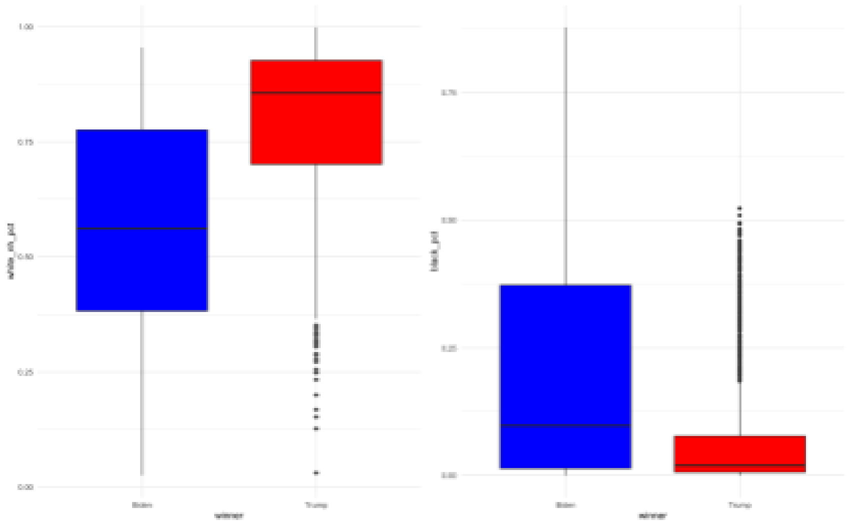


Figure 5: Proportion of white vs. black voters by winner

Furthermore, in Figure 5 we can see differences in voting behavior by ethnicity. Counties Trump won tend to be substantially whiter, with tight spread. Biden counties are seen to have lower white shares and notably higher black population percentages. Different ethnic/racial groups differ on average and these differences are key to shaping who won.

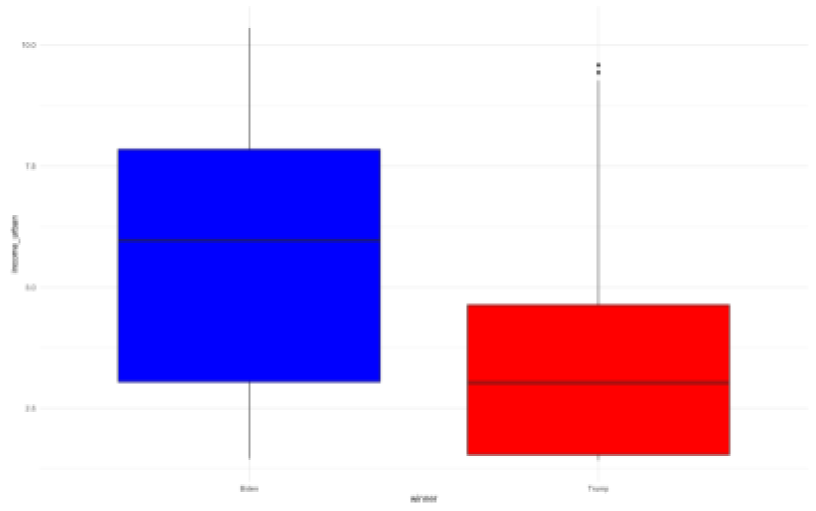


Figure 6: Urban-weighted income by winner

Even further, we can combine variables to capture more nuanced patterns. For example, weighting income by urbanity as seen in figure 6 shows Biden counties show much higher urban-weighted income while Trump counties are lower with fewer high-income urban outliers. This composite process captures joint effects better than raw income or urban share alone. In our preprocessing we take thoughtful transformations like proportions and weighted composites to make comparisons across counties fairer and more representative of relative differences.

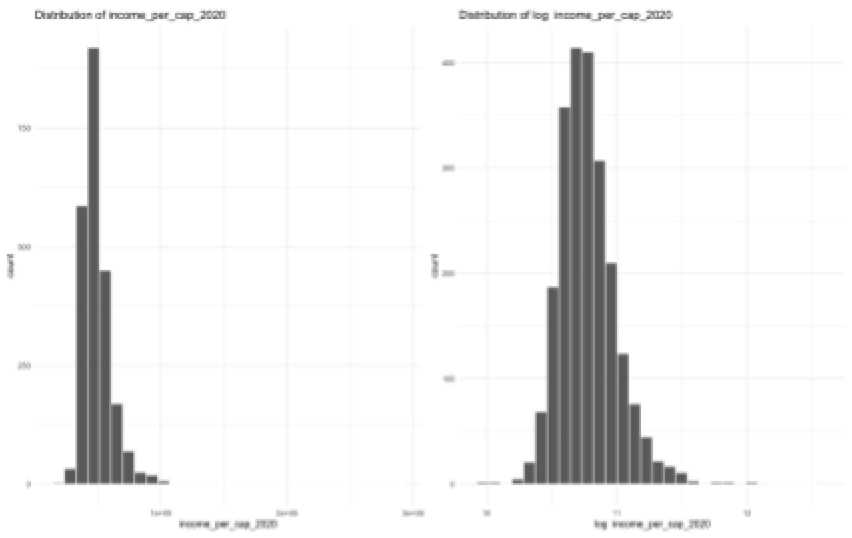


Figure 7: Effect of transforming income per capita 2020 to log on distribution

In Figure 7, income_per_cap_2020 is right skewed, while the log view looks much closer to symmetric. It may be useful to transform this feature with a log when values are strictly positive or Yeo Johnson when zeros appear, followed by normalization. The same approach can be applied to other variables that show similar skewed patterns to reduce skewness, stabilize variance, and make comparisons across counties more meaningful.

It may also be useful to treat certain variables as categorical. For example, possibly converting the CDC urban–rural code (x2013_code) to a factor, which runs from 1 to 6 with 1 being most urban and 6 being most rural, allows us to treat the variable as a category which is more representative of the variable in some cases. Likewise, ensuring the winner is stored as a factor allows models to properly handle class membership.

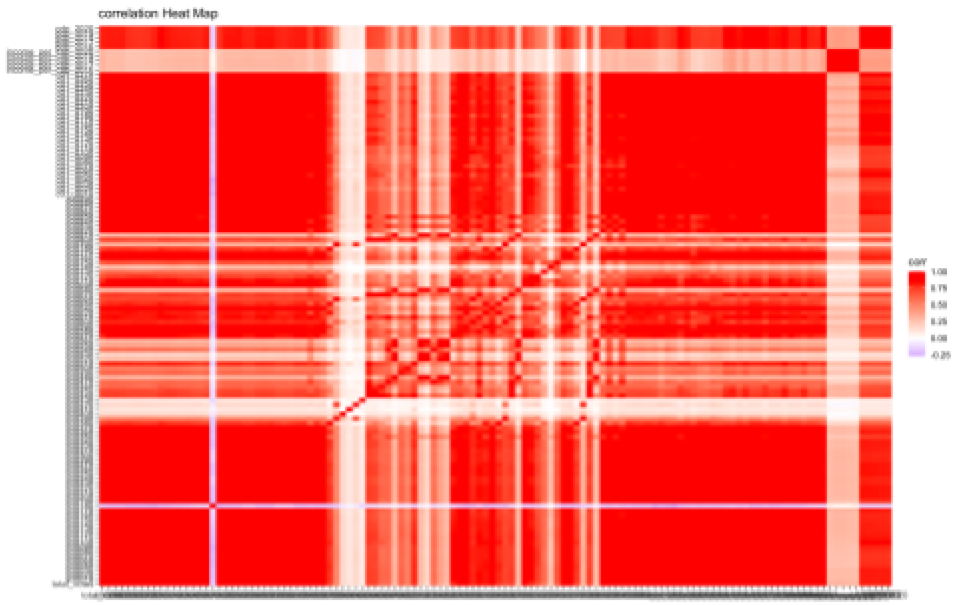


Figure 8: Heat Map

Examining Figure 8, the dense red blocks across the visual reveal strong collinearity that can destabilize coefficient-based models. In preprocessing, we look to reduce the risk of multicolinearity through carefully utilizing transformations discussed in above sections and adding targeted interactions, such as income multiplied by urban share, to capture joint effects.

Overall, our exploratory data analysis supports using log or Yeo–Johnson transformations for skewed variables, normalization to stabilize scale, proportions to adjust for county size, factor encoding for categorical variables, and targeted interactions. These steps aim to improve interpretability and model stability while reducing the risk of multicolinearity.

Recipes and Preprocessing

In preparation for modeling the 2020 U.S. Presidential Election outcome at the county level, we applied a series of preprocessing and feature engineering steps to both training and test datasets. Our preprocessing approach was informed by the exploratory data analysis, which highlighted issues such as skewed numeric distributions, strong collinearity among predictors, and class imbalance favoring Trump counties.

The target variable, *winner*, was converted to a factor to ensure proper handling of class membership across classification models. Redundant or identifying columns such as *id* and *name* were removed from the predictors, while *id* was retained in the test set solely for submission purposes.

Across models, those following are common preprocessing steps were applied:

1. Numeric predictors with missing values, particularly income and GDP estimates, were imputed using the median to reduce bias from extreme values.
2. Skewed numeric variables were transformed using the Yeo-Johnson procedure to allow for zero and negative values. All numeric predictors were subsequently normalized to have mean zero and until variance (*step_normalize*), ensuring comparability across features and stabilizing models sensitive to scale.
3. Features with near-zero variance were removed to prevent noise from non-informative variables.
4. To account for differences in county size, several predictors were expressed as proportions, e.g., demographic groups over total population, turnout rate, and urban-weighted income. Composite variables, such as income growth and GDP per capita, were computed to capture temporal and structural effects.

While the above preprocessing steps were shared across models, we added additional specialized transformations to several models.

- K-Nearest Neighbors (KNN): Standardized all numeric predictors after imputation to make distances comparable; otherwise used the shared engineered features and missingness indicators.
- XGBoost: no standardization; retained shared features and indicators, added two interactions (bachelor's-degree share × urban level; bachelor's-degree share × turnout), and dropped helper denominators.
- Naive Bayes: kernel NB with continuous inputs; no discretization or standardization; kept the shared features and indicators, removing only temporary denominators.

Candidate Models / Model Evaluation

We evaluated seven candidate models to predict the winner of each county in the 2020 U.S. Presidential Election. The baseline models included Logistic Regression, Linear Discriminant Analysis (LDA), and Quadratic Discriminant Analysis (QDA). To capture more complex decision boundaries and potential nonlinearities, we additionally tested Support Vector Machines (SVM), K-Nearest Neighbors (KNN), XGBoost, and Navie Bayes.

Model descriptions:

1. **Model 1 (Logistic Regression):** Logistic regression provides a simple, interpretable baseline that can be regularized with lasso or ridge penalties to handle multicollinearity.
2. **Model 2 (LDA):** Linear Discriminant Analysis assumes equal covariance across classes, offering a lower-variance but less flexible classifier.
3. **Model 3 (QDA):** Quadratic Discriminant Analysis relaxes the equal covariance assumption, increasing flexibility but at the cost of higher variance.
4. **Model 4 (SVM):** Support Vector Machines with an RBF kernel capture nonlinear decision boundaries, tuned via Bayesian optimization.
5. **Model 5 (KNN):**A KNN classification model fit with kknn on standardized predictors; k evaluated at 10 levels from 5–50, distance power set to {1 = Manhattan, 2 = Euclidean}, and kernel weights chosen from {rectangular, triangular, epanechnikov, biweight, triweight, cos, inv, gaussian}; best setting selected by stratified 5-fold CV on accuracy.
6. **Model 6 (XGBoost):** An XGBoost classifier trained on engineered features with ranges: number of trees = 600–1600, maximum tree depth = 3–10, minimum samples per leaf =

5–30, minimum loss reduction to split = 0.01–1, row subsampling rate = 0.60–0.90, and learning rate = 0.003–0.032; objective = binary:logistic, evaluation metric = logloss, column subsampling by tree/level = 0.8; best setting chosen by stratified 5-fold CV on ROC AUC.

7. **Model 7 (Naive Bayes):** A kernel Naive Bayes classifier fit with naivebayes (usekernel = TRUE) using smoothness = 0.8–1.6 and Laplace = 1–2; tuned with stratified 5×2 CV and selected by minimum multinomial log loss.

Candidate Models Summary Table

The table below summarizes the candidate models, including their type, engine, preprocessing or features used, and the key hyperparameters for each approach.

Model #	**Model Type**	**Engine**	**Recipe / Variable Used**	**Hyperparameters**
Model 1	Logistic Regression	glm	Preprocessed recipe	-
Model 2	LDA	MASS	Preprocessed recipe	-
Model 3	QDA	MASS	Preprocessed recipe	-
Model 4	SVM	kernlab	Preprocessed recipe	Cost, σ tuned via Bayesian optimization
Model 5	KNN	kknn	demographic & economic ratios; growth; urbanicity; education shares;	neighbors; distance metric; weighting kernel

			diversity; missingness; standardized numerics	
Model 6	XGBoost	xgboost	same engineered features as above; + two interactions; no standardization	number of trees, max depth, learning rate, row subsample, column subsample, min loss to split, min samples per leaf
Model 7	Naive Bayes	naivebayes	same engineered features as above; kernel densities; no discretization/standa rdization	smoothness, laplace smoothing, kernel density

Model Evaluation and Tuning

We used 5-fold cross-validation to assess the performance of all candidate models, ensuring each model was tested on multiple subsets of the data for robust evaluation. Accuracy was chosen as the primary metric, while ROC AUC was additionally reported to evaluate performance under class imbalance (Trump counties ≈ 83%, Biden counties ≈ 17 %)

Hyperparameters were tuned as follows:

- Logistic Regression, LDA, QDA - default specifications as baseline linear models.
- SVM - cost and RBF kernel width tuned via Bayesian optimization

- KNN - neighbors, distance metric, and kernel weighting tuned across a wide grid (k = 5-50)
- XGBoost - number of trees, maximum depth, learning rate, row/column subsampling, and minimum loss reduction tuned via cross-validation.
- Naive Bayes - smoothness and Laplace parameters tuned with repeated 5-fold CV.

Model #	**CV Accuracy**	**ROC AUC**	**ROC AUC Std. Error**
Model 1 (Logistic)	0.934	0.961	0.00380
Model 2 (LDA)	0.925	0.958	0.00426
Model 3 (QDA)	0.905	0.917	0.00884
Model 4 (SVM)	0.939	0.966	0.00362
Model 5 (KNN)	0.916	0.944	0.00651
Model 6 (XGBoost)	0.939	0.973	0.00226
Model 7 (Naive Bayes)	0.835	0.796	0.00898

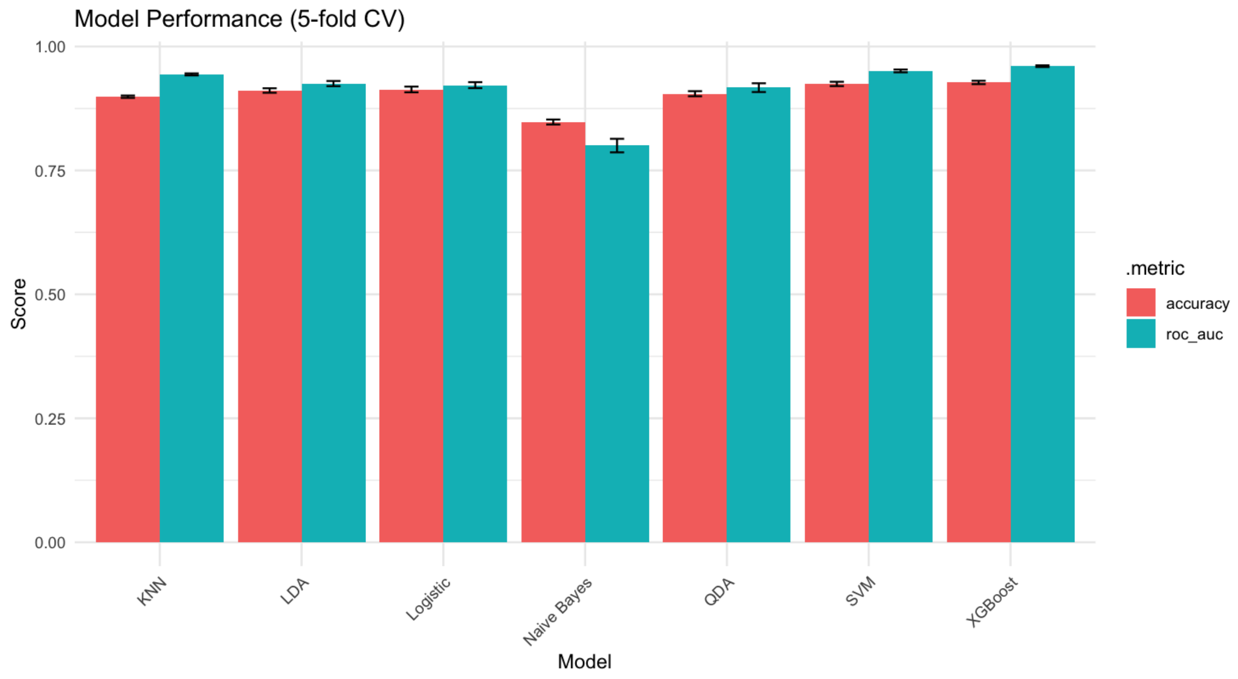
Discussion of Results

Among the baseline linear models, Logistic Regression achieved high accuracy (0.934) and strong ROC AUC (0.961), demonstrating that a simple linear boundary captures most of the country level voting patterns. LDA performed similarly but slightly lower in both accuracy and ROC AUC, reflecting its lower flexibility due to the equal covariance assumption. QDA demonstrated reduced performance due to overfitting in this high dimensional setting with imbalance classes.

Nonlinear models showed modest improvements. SVM achieved the highest accuracy among non-ensemble methods (0.939) and slightly improved ROC AUC (0.966), indicating that modeling nonlinear boundaries captures additional patterns beyond linear separation.
KNN performed reasonably well but was less stable, with slightly lower accuracy and ROC AUC compared to SVM, suggesting sensitivity to the choice of k and distance metrics in this imbalance dataset.

Ensemble and boosting methods showed the strongest performance. XGBoost matched SVM in accuracy (0.939) but achieved the highest ROC AUC (0.973) and lowest standard error, highlighting its ability to capture complex interactions and handle feature heterogeneity effectively. Despite its simplicity, Naive Bayes underperformed relative to all other models, reflecting limitations in modeling complex conditional dependencies and handling class imbalance.

Overall, XGBoost stands out as the best performing model, balancing high accuracy with strong discrimination as measured by ROC AUC, while linear and discriminant models remain competitive as interpretable baselines. The comparison plot below summarizes accuracy and ROC AUC across all candidate models.



Discussion of final model

We selected XGBoost for its strong and reliable cross-validated performance. It consistently outperformed simpler baselines in discrimination while maintaining stable variance across folds. A key advantage was its generation of well-calibrated class probabilities. This was essential for tuning a decision threshold to optimize for our target metric, rather than being limited to a fixed cutoff.

XGBoost is also a natural fit for county-level data. The model effectively captures nonlinearities and interactions without extensive manual feature engineering, simplifying our preprocessing pipeline. Its robustness to correlated inputs and varied data scales translated directly into consistently higher AUC and accuracy.

However, the choice of XGBoost involves clear trade-offs. Model interpretability is less direct, necessitating post-hoc tools to explain feature contributions. Standard cross-validation might also produce optimistic estimates if spatial dependence exists between counties. Finally, we must remain mindful of practical risks, such as probability calibration drifting on new data and potential information leakage from contemporaneous features.

Several opportunities exist to strengthen the classifier, focusing on three key areas.

First, we can adopt a more robust validation and tuning framework. Shifting to a grouped or spatial cross-validation by state would better simulate real-world deployment. Within this framework, the tuning process can be enhanced by implementing early stopping to prevent overfitting.

Second, a deeper analysis of model behavior would allow for targeted refinements. A detailed error analysis across geographic strata is essential for identifying systematic failures. To address potential instability or miscalibration, we could also explore a lightweight ensemble, pairing the booster with a penalized logistic regression baseline to improve overall reliability.

Finally, enriching the feature set with more diverse data sources would likely yield significant performance gains. Beyond demographics, we can incorporate features that capture a county's political context and socio-economic landscape. Data on historical election results, economic activity, infrastructure, and community structure would provide powerful predictive signals.

These additions would not only improve accuracy but also help the model generalize by reducing the risk of year-specific data leakage.

Appendix: Final annotated script

```
#loading tidy models and ggplot
library(tidymodels)
library(ggplot2)
#patchwork allows us to put multiple ggplot visuals together as
one
#did this bc par does not work for ggplot but have visuals side
by side looks clean
library(patchwork)
#libraries for models
library(car)
library(ISLR)
library(MASS)
# Additional libraries for new models
library(discrim)
library(naivebayes)
library(xgboost)

tidymodels_prefer()
#were gonna use large comment chunks to break up the sections of
our script!

#================================================================
==============
#                                 LOADING DATA
#================================================================
==============

#loading training and testing sets
train <- read.csv("train_class.csv")
test  <- read.csv("test_class.csv")


#================================================================
==============
#                                 eDA SecTION
#================================================================
==============

###############################################
##### variable Distributions section ##########
###############################################
#in this section we are just trying to get a feel for how the
variables are distributed
```

```
#trying to see the skewness of the variables and what not
#also trying to see the trends of distributions

#proportions of biden vs trump
table(train$winner)
prop.table(table(train$winner))

#dist of outcome
ggplot(train, aes(x = winner)) +
  geom_bar() +
  labs(title = "Distribution of Outcome class: Winner",
       x = "Winner", y = "count") +
  theme_minimal()


###########################
######## count vars #######
###########################

#creating a funcition that inputs our data set (train.csv) and
the variables for optimal plotting
#also allows less repitition!
#the ggplot function creates a histogram of the specified
variable!
plot_hist <- function(var, data) {
  ggplot(data, aes(x = .data[[var]])) +
    geom_histogram(bins = 30, color = "white", alpha = 0.9) +
    labs(
      title = paste("Distribution of", var),
      x = var,
      y = "count"
    ) +
    theme_minimal()
}

log_plot_hist <- function(var, data) {
  ggplot(data, aes(x = log(.data[[var]]))) +
    geom_histogram(bins = 30, color = "white", alpha = 0.9) +
    labs(
      title = paste("Distribution of log ", var),
      x = paste("log ", var),
      y = "count"
    ) +
    theme_minimal()
}
#the variables will be all the predictors so in vars omitting
order total
#we need the names of the columns to be able to use the function
so use names()
vars <- train %>%
  names()


#generate histogram plots using lapply
dist_plots <- lapply(vars, plot_hist, data = train)
log_dist_plots <- lapply(vars, log_plot_hist, data = train)

####################
# tot pop/ gender  #
####################
dist_plots[[6]]

####################
# income          #
####################
dist_plots[[120]]|log_dist_plots[[120]]


###############################################
##### Relationiships w/ outcome sect ##########
###############################################
#############
#categorical#
#############
#gonna use bar charts
#allows us to see the relationships of categorical variables
with outcome
total_pop <- ggplot(train, aes(x = winner, y = x0001e, fill =
winner)) +
  geom_boxplot() +
  labs(
    title = "Boxplot of total population by Outcome class
(Winner)",
    x = "Winner",
    y = "total_population"
  ) +
  theme_minimal() +
  theme(legend.position = "none")+
  scale_fill_manual(values = c("Biden" = "blue", "Trump" =
"red"))

total_pop


#gender proportion
male_share <- train$x0002e / (train$x0001e + 1e-9)
female_share <- train$x0003e / (train$x0001e + 1e-9)
female_proportion_plot <- ggplot(train, aes(winner,
female_share, fill = winner)) +
  geom_boxplot() +
  theme_minimal()+
  theme(legend.position = "none")+
  scale_fill_manual(values = c("Biden" = "blue", "Trump" =
"red"))
male_proportion_plot <- ggplot(train, aes(winner, male_share,
fill = winner)) +
  geom_boxplot() +
  theme_minimal()+
  theme(legend.position = "none")+
  scale_fill_manual(values = c("Biden" = "blue", "Trump" =
"red"))
male_proportion_plot | female_proportion_plot

#white vs black proportion
white_nh_pct = train$x0077e / (train$x0001e + 1)
black_pct = train$x0038e / (train$x0001e + 1)
white_proportion_plot <- ggplot(train, aes(winner, white_nh_pct,
fill = winner)) +
  geom_boxplot() +
  theme_minimal()+
  theme(legend.position = "none")+
  scale_fill_manual(values = c("Biden" = "blue", "Trump" =
"red"))
black_proportion_plot <- ggplot(train, aes(winner, black_pct,
fill = winner)) +
  geom_boxplot() +
  theme_minimal()+
  theme(legend.position = "none")+
  scale_fill_manual(values = c("Biden" = "blue", "Trump" =
"red"))
white_proportion_plot | black_proportion_plot
```

```
#income and urban
urban_level = 7 - train$x2013_code
income_urban = log(train$income_per_cap_2020 + 1) * urban_level
/ 7
income_urban_plot <- ggplot(train, aes(winner, income_urban,
fill = winner)) +
  geom_boxplot() +
  theme_minimal()+
  theme(legend.position = "none")+
  scale_fill_manual(values = c("Biden" = "blue", "Trump" =
"red"))
income_urban_plot


##############################
######## TRANSFORMATIONS #####
##############################
#log and step tranformations

################################
## heat map                  ##
################################
numeric_for_heat_map <- train %>%
  select(-id, -name, -x2013_code, -winner)

#correlation matrix
cormat <- cor(numeric_for_heat_map, use =
"pairwise.complete.obs")
df <- as.data.frame(as.table(cormat))
names(df) <- c("var1", "var2", "corr")

#heat map in ggplot
ggplot(df, aes(x = var2, y = var1, fill = corr)) +
  geom_tile() +
  scale_fill_gradient2(low="blue", mid="white", high="red",
midpoint=0) +
  labs(x = NULL, y = NULL, title = "correlation Heat Map")



#==============================================================
==============
#                                PRe PROceSSING
```

```
#================================================================
==============
#winner is factor
train$winner <- factor(train$winner)
train_clean <- train %>% select(-name)

#recipe with preprocessing
# Recipe with preprocessing
rec <- recipe(winner ~ ., data = train_clean) %>%
  update_role(id, new_role = "ID") %>%
  #imppute before mutate
  step_impute_median(all_numeric())%>%
  step_mutate(
    log_income = log(income_per_cap_2020 + 1),
    male_vote = x0002e / (x0001e + 1),
    female_vote = x0003e / (x0001e + 1),
    income_growth_4yr = (income_per_cap_2020 -
income_per_cap_2016) / (income_per_cap_2016 + 1),
    income_growth_1yr = (income_per_cap_2020 -
income_per_cap_2019) / (income_per_cap_2019 + 1),
    gdp_per_capita = gdp_2020 / (x0001e + 1),
    gdp_growth = (gdp_2020 - gdp_2016) / (gdp_2016 + 1),

    turnout_rate = total_votes / (x0001e + 1),
    adult_turnout = total_votes / (x0001e - x0005e - x0006e -
x0007e + 1),

    white_nh_pct = x0077e / (x0001e + 1),
    black_pct = x0038e / (x0001e + 1),
    hispanic_pct = x0071e / (x0001e + 1),
    asian_pct = x0044e / (x0001e + 1),
    minority_pct = 1 - (x0077e / (x0001e + 1)),

    urban_level = 7 - x2013_code,
    is_large_metro = ifelse(x2013_code <= 2, 1, 0),
    is_rural = ifelse(x2013_code >= 5, 1, 0),

    youth_pct = (x0008e + x0009e + x0010e) / (x0001e + 1),
    working_age_pct = (x0011e + x0012e + x0013e + x0014e) /
(x0001e + 1),
    senior_pct = (x0015e + x0016e + x0017e) / (x0001e + 1),

    log_population = log(x0001e + 1),
    diversity = 1 - ((x0037e/(x0001e+1))^2 +
(x0038e/(x0001e+1))^2 +
                       (x0044e/(x0001e+1))^2),

    white_rural = white_nh_pct * is_rural,
    minority_urban = minority_pct * (1 - is_rural),
    income_urban = log_income * urban_level / 7,
    turnout_senior = turnout_rate * senior_pct

  ) %>%
  step_rm(-any_of(c("id", "winner", "white_nh_pct", "black_pct",
"hispanic_pct",
                    "asian_pct", "minority_pct", "youth_pct",
"working_age_pct",
                    "senior_pct", "turnout_rate",
"adult_turnout", "log_income",
                    "income_growth_4yr", "income_growth_1yr",
"gdp_per_capita",
                    "gdp_growth", "log_population",
"urban_level", "is_large_metro",
                    "is_rural", "diversity", "white_rural",
"minority_urban",
                    "income_urban", "turnout_senior"))) %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())

qda_rec <- recipe(winner ~ ., data = train_clean) %>%
  update_role(id, new_role = "ID") %>%
  step_impute_median(all_numeric()) %>%
  step_mutate(
    log_income = log(income_per_cap_2020 + 1),
    white_nh_pct = x0077e / (x0001e + 1),
    turnout_rate = total_votes / (x0001e + 1),
    urban_level = 7 - x2013_code,
    log_population = log(x0001e + 1)
  ) %>%
  step_rm(-any_of(c("id", "winner", "log_income",
"white_nh_pct", "turnout_rate", "urban_level",
"log_population"))) %>%
  step_normalize(all_numeric_predictors())

# Advanced recipe for KNN, Naive Bayes, and XGBoost
advanced_rec <- recipe(winner ~ ., data = train_clean) %>%
  update_role(id, new_role = "ID") %>%
  step_mutate(
    # size & economy
    log_population    = log(x0001e + 1),
    gdp_per_capita    = gdp_2020 / (x0001e + 1),
    log_gdp_pc        = log(gdp_per_capita + 1e-6),
    gdp_growth        = (gdp_2020 - gdp_2016) / (gdp_2016 + 1),

    # income level & growth
    log_income        = log(income_per_cap_2020 + 1),
    income_growth_4yr = (income_per_cap_2020 -
income_per_cap_2016) / (income_per_cap_2016 + 1),
    income_growth_1yr = (income_per_cap_2020 -
income_per_cap_2019) / (income_per_cap_2019 + 1),

    # racial composition
    white_nh_pct = x0077e / (x0001e + 1),
    black_pct    = x0038e / (x0001e + 1),
    hispanic_pct = x0071e / (x0001e + 1),
    asian_pct    = x0044e / (x0001e + 1),
    minority_pct = 1 - (x0077e / (x0001e + 1)),

    # age composition
    age18_24_pct  = c01_001e / (x0001e + 1),
    age25_34_pct  = c01_016e / (x0001e + 1),
    age35_44_pct  = c01_019e / (x0001e + 1),
    age45_64_pct  = c01_022e / (x0001e + 1),
    age65plus_pct = c01_025e / (x0001e + 1),

    # turnout rate
    turnout_rate   = total_votes / (x0001e + 1),

    # urbanicity
    urban_level    = 7 - x2013_code,
    is_large_metro = ifelse(x2013_code <= 2, 1, 0),
    is_rural       = ifelse(x2013_code >= 5, 1, 0),

    # education denominator
    edu_25plus = pmax(c01_006e, 1),

    # education composition
    hs_plus_pct     = c01_014e / edu_25plus,
    bach_plus_pct   = c01_015e / edu_25plus,
    grad_degree_pct = c01_013e / edu_25plus,
    # diversity index
    diversity = 1 - (white_nh_pct^2 + black_pct^2 +
hispanic_pct^2 + asian_pct^2)
  ) %>%
  # remove helpers
  step_rm(edu_25plus, gdp_per_capita) %>%
  # missingness indicators
  step_indicate_na(
    income_per_cap_2016, income_per_cap_2017,
income_per_cap_2018,
    income_per_cap_2019, income_per_cap_2020,
    gdp_2016, gdp_2017, gdp_2018, gdp_2019, gdp_2020
  ) %>%
  # interactions
  step_interact(~ bach_plus_pct:urban_level +
bach_plus_pct:turnout_rate) %>%
  step_impute_median(all_numeric_predictors()) %>%
  step_nzv(all_predictors()) %>%
  step_normalize(all_numeric_predictors())

#================================================================
==============
#                                 MODeL cReATION
#================================================================
==============

######################
###  Logistic       ##
######################
#can apply lasso or ridge via penalty argument
logistic_mod <- logistic_reg(mode = "classification") %>%
  set_engine("glm")

logistic_wf <- workflow() %>%
  add_recipe(rec) %>%
  add_model(logistic_mod)

logistic_fit <- logistic_wf %>%
  fit(data = train_clean)

test_preds_logistic <- predict(logistic_fit, new_data = test)

##################################
## linear discriminant analysis ##
```

```
##################################
#less flexible, more bias, less variance
lda_mod <- discrim_linear() %>%
  set_engine("MASS") %>%
  set_mode("classification")

lda_wf <- workflow() %>%
  add_recipe(rec) %>%
  add_model(lda_mod)

lda_fit <- lda_wf %>%
  fit(data = train_clean)

test_preds_lda <- predict(lda_fit, new_data = test)

#####################################
## Quadratic discriminant analysis ##
#####################################
#more flexible, less bias, more variance
qda_mod <-
  discrim_quad(mode = "classification") %>%
  set_engine("MASS")

qda_wf <- workflow() %>%
  add_recipe(qda_rec) %>%
  add_model(qda_mod)

qda_fit <-
  qda_wf %>%
  fit(data = train_clean)

test_preds_qda<- predict(qda_fit, new_data = test)

######################
###  SVM           ##
######################
#specify svm model
svm_model <-
  svm_rbf(cost = tune(),
          rbf_sigma = tune()) %>%
  set_engine("kernlab") %>%
  set_mode("classification")

#set svm workflow
```

```
svm_wf <-
  workflow() %>%
  add_recipe(rec) %>%
  add_model(svm_model)

######################
###  KNN           ##
######################
knn_spec <- nearest_neighbor(
  neighbors   = tune(),
  weight_func = tune(),
  dist_power  = tune()
) %>%
  set_engine("kknn") %>%
  set_mode("classification")

knn_wf <- workflow() %>%
  add_recipe(advanced_rec) %>%
  add_model(knn_spec)

######################
###  Naive Bayes    ##
######################
nb_spec <- naive_Bayes(
  smoothness = tune(),
  Laplace    = tune()
) %>%
  set_engine("naivebayes", usekernel = TRUE) %>%
  set_mode("classification")

nb_wf <- workflow() %>%
  add_recipe(advanced_rec) %>%
  add_model(nb_spec)

######################
###  XGBoost        ##
######################
xgb_spec <- boost_tree(
  trees          = tune(),
  tree_depth     = tune(),
  min_n          = tune(),
  loss_reduction = tune(),
  sample_size    = tune(),
  learn_rate     = tune()
) %>%
  set_engine(
    "xgboost",
    objective = "binary:logistic",
    eval_metric = "logloss",
    counts = FALSE,
    colsample_bytree  = 0.8,
    colsample_bylevel = 0.8
  ) %>%
  set_mode("classification")

xgb_wf <- workflow() %>%
  add_recipe(advanced_rec) %>%
  add_model(xgb_spec)

#================================================================
==============
#                            MODEL TUNING & CROSS VALIDATION
#================================================================
==============

# cross validation
set.seed(123)
cell_folds <- vfold_cv(train_clean, v = 5, strata = winner)

#roc metrics
roc_res <- metric_set(roc_auc, accuracy)

# Fit models without tuning first
res_log <- fit_resamples(logistic_wf, resamples = cell_folds,
metrics = roc_res)
res_lda <- fit_resamples(lda_wf, resamples = cell_folds, metrics
= roc_res)
res_qda <- fit_resamples(qda_wf, resamples = cell_folds, metrics
= roc_res)

# SVM tuning
svm_param <- svm_wf %>%
  extract_parameter_set_dials() %>%
  update(
    cost = cost(range = c(-6, 2), trans = scales::log2_trans()),
    rbf_sigma = rbf_sigma(range = c(-8, 0), trans =
scales::log2_trans())
  )
```

```
start_grid <- grid_latin_hypercube(svm_param, size = 12)

svm_initial <- tune_grid(
  svm_wf,
  resamples = cell_folds,
  grid = start_grid,
  metrics = roc_res
)

ctrl <- control_bayes(verbose = TRUE)

set.seed(123)
svm_bo <- tune_bayes(
  svm_wf,
  resamples  = cell_folds,
  metrics    = roc_res,
  param_info = svm_param,
  initial    = svm_initial,
  iter       = 25,
  control    = ctrl
)

# KNN tuning
knn_grid <- grid_regular(
  neighbors(range = c(5, 50)),
  weight_func(values =
c("rectangular","triangular","epanechnikov",

"biweight","triweight","cos","inv","gaussian")),
  dist_power(range = c(1, 2)),
  levels = c(10, 8, 2)
)

knn_tuned <- tune_grid(
  knn_wf,
  resamples = cell_folds,
  grid = knn_grid,
  metrics = roc_res,
  control = control_grid(save_pred = TRUE, verbose = TRUE)
)

# Naive Bayes tuning
nb_grid <- grid_regular(
  smoothness(range = c(0.8, 1.6)),
  Laplace(range = c(1, 2)),
  levels = c(4, 2)
)

nb_tuned <- tune_grid(
  nb_wf,
  resamples = cell_folds,
  grid = nb_grid,
  metrics = metric_set(accuracy, roc_auc, mn_log_loss),
  control = control_grid(save_pred = TRUE, verbose = TRUE)
)

# XGBoost tuning
xgb_grid <- grid_latin_hypercube(
  trees(range = c(600, 1600)),
  tree_depth(range = c(3, 10)),
  min_n(range = c(5, 30)),
  loss_reduction(range = c(-2, 0)),
  sample_size = sample_prop(range = c(0.6, 0.9)),
  learn_rate(range = c(-2.5, -1.5)),
  size = 36
)

xgb_tuned <- tune_grid(
  xgb_wf,
  resamples = cell_folds,
  grid = xgb_grid,
  metrics = roc_res
)

#================================================================
==============
#                            FINAL MODEL FITTING
#================================================================
==============

# Get best parameters and fit final models
best_svm <- select_best(svm_bo, metric = "roc_auc")
final_svm_wf <- finalize_workflow(svm_wf, best_svm)
final_svm_fit <- fit(final_svm_wf, data = train_clean)
test_preds_svm <- predict(final_svm_fit, new_data = test)

best_knn <- select_best(knn_tuned, metric = "accuracy")
```

```
final_knn_wf <- finalize_workflow(knn_wf, best_knn)
final_knn_fit <- fit(final_knn_wf, data = train_clean)
test_preds_knn <- predict(final_knn_fit, new_data = test)

best_nb <- select_best(nb_tuned, metric = "mn_log_loss")
final_nb_wf <- finalize_workflow(nb_wf, best_nb)
final_nb_fit <- fit(final_nb_wf, data = train_clean)
test_preds_nb <- predict(final_nb_fit, new_data = test)

best_xgb <- select_best(xgb_tuned, metric = "roc_auc")
final_xgb_wf <- finalize_workflow(xgb_wf, best_xgb)
final_xgb_fit <- fit(final_xgb_wf, data = train_clean)
test_preds_xgb <- predict(final_xgb_fit, new_data = test)

#================================================================
==============
#                            MODEL COMPARISON
#================================================================
==============

# Collect all results for comparison
all_results <- bind_rows(
  collect_metrics(res_log) %>% mutate(model = "Logistic"),
  collect_metrics(res_lda) %>% mutate(model = "LDA"),
  collect_metrics(res_qda) %>% mutate(model = "QDA"),
  show_best(svm_bo, metric = "roc_auc", n = 1) %>% mutate(model
= "SVM"),
  show_best(knn_tuned, metric = "accuracy", n = 1) %>%
mutate(model = "KNN"),
  show_best(nb_tuned, metric = "roc_auc", n = 1) %>%
mutate(model = "Naive Bayes"),
  show_best(xgb_tuned, metric = "roc_auc", n = 1) %>%
mutate(model = "XGBoost")
)

print(all_results)

#================================================================
==============
#                                WRITe TO cSV
#================================================================
==============
# Choose the best model for final submission (you can change
this based on results)
final_preds <- test_preds_logistic %>%
  bind_cols(test %>% select(id)) %>%
  rename(winner = .pred_class) %>%
  select(id, winner)

write_csv(final_preds, "submission.csv")

# Write individual model predictions for comparison
test_preds_svm %>% bind_cols(test %>% select(id)) %>%
  rename(winner = .pred_class) %>%
  select(id, winner) %>%
  write_csv("svm_submission.csv")

test_preds_knn %>% bind_cols(test %>% select(id)) %>%
  rename(winner = .pred_class) %>%
  select(id, winner) %>%
  write_csv("knn_submission.csv")

test_preds_nb %>% bind_cols(test %>% select(id)) %>%
  rename(winner = .pred_class) %>%
  select(id, winner) %>%
  write_csv("nb_submission.csv")

test_preds_xgb %>% bind_cols(test %>% select(id)) %>%
  rename(winner = .pred_class) %>%
  select(id, winner) %>%
  write_csv("xgb_submission.csv")
```

Appendix: Team member contributions

Claire Nabours: logistic, linear discriminant analysis, quadratic discriminant analysis, and svm models with bayesian optimization and preprocessing for those models, EDA section of report.

Jackie Sung: Recipes and preprocessing section of report, Candidate Models Summary Table + Model Evaluation and Tuning + discussion of result of report

Naikang Wang: KNN, Naive Bayes, XGBoost, Introduction, Discussion of final model

Yankai Wen: …