B9DA110 Advanced Data and Network Mining: CA_TWO

Cross-Industry Standard Process – Data Mining

Submitted by:
Anish Rao: 20066423

Lecturer: Oleksandr Bezrukavyi

Table of Contents

1. Introduction *2*

2. Methodology *3*

2.1 Business Understanding 3

2.2 Data Understanding 3

2.3 Data Preparation 5

2.4 Modelling *6*

2.4.1 RapidMiner AutoModel 6

2.4.2 Python 7

3. Evaluation *9*

4. Model Deployment *10*

5. Conclusion *11*

References *12*

Appendix A – RapidMiner *12*

Appendix B – Python *17*

1. Introduction

The Cross-Industry Standard Process for Data Mining (CRISP-DM) is a method that is very commonly used because it gives a structure for data mining projects. It was created to give models that can be used in any industry and with any kind of technology. It makes sure that data driven projects are similar and can be replicated on different types of organizations and different data driven initiatives (Wirth & Hipp, 2000).

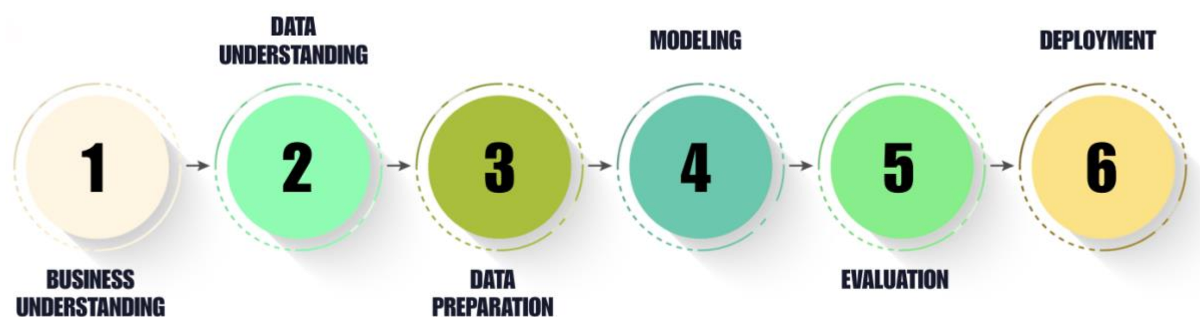


Figure 1: CRISP-DM phases

The whole process is a cycle of six different but interrelated components, Business Understanding, Data Understanding, Data Preparation, Modelling, Evaluation, and Deployment. This kind of structure gives flexibility to adjust the process according to the organization but also offering a proper logical framework. Each stage depends on the previous stages but it also makes it possible for making small changes during the full process.

A very important use case for machine learning is income prediction. It has lots of impact in a few major industries. Financial institutions use income prediction models when they are deciding to give credit, assess risk when giving loan or decide how to properly target people for their premium products. These models can also be used by government agencies to check if a person is eligible for any social programs or how to a allocate resources. It can also help in marketing as they can segment customers and do targeted promotional campaigns.

The Adult Census Income dataset (Becker & Kohavi, 1996) is a very good dataset to demonstrate how useful CRISP-DM can be. The dataset has demographic, educational, and employment information for 48,842 individuals and predict if their annual income exceeds 50,000. The dataset also has issues like missing data, class imbalance etc. like in real world.

This project uses a dual-platform approach where our models are implemented using python programming and the AutoModel feature of RapidMiner. Python makes it easy to see and manage each step of the model giving us full control. RapidMiner provides fully automated process which doesn’t need any manual coding like in python and provide better results also.

Checking the differences between manual and automated approaches is very useful for assessing the current practices in data science. It is also important to understand how better or worse automated tools can be as it helps in deciding which tool is best for specific business case.

2. Methodology

2.1 Business Understanding

Income prediction is needed in industries where monetary precision is very important for corporate decisions. In the banking industry predicting income is important for credit approval process, deciding the loan amount and managing risks. It is also important for insurance companies for deciding their policies and premiums. Government and political departments also rely on income classification for targeting social programs and deciding on policies.

The business objective is to design a system that can accurately predict if an individual is earning more or less than $50,000 annually. This number is important predicting more or less than this figure will help organizations to better assess customer targeting, risk management and resource allocation.

Stakeholder Analysis: The main stakeholders are financial institutions looking for optimised credit decisions, government agencies looking for efficient policy development and marketing teams looking to create targeted campaigns. Secondary stakeholders include insurance firms performing risk assessment, HR teams who might want to decide salaries and economists who need information to create economic policies.

The technical objectives in our case is building a binary classification model that decides if a person will have income more or less than $50,000, based on the demographic and employment details we have. The results of different models will be defined using metrics like accuracy, precision, recall (the most important for our case) and F1-score. We also need to focus on balanced results to make sure the models work well in real world.

The evaluation method also differs according to the business context. In financial services the focus is more on recall to avoid missing high earners. While in marketing precision is more preferred to waste any extra efforts in marketing. Understanding what our business goal is therefore very important on model selection and optimisation.

2.2 Data Understanding

The Adult Census Income dataset is available on the UCI Machine Learning Repository (Becker & Kohavi, 1996). The dataset has 48,842 entries with 14 features that are demographic, employment, educational and financial types of data. Each record is for a person from the 1994 US Census. The target variable in the dataset is whether annual income exceeds $50,000.

```
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             48842 non-null  int64 
 1   workclass       47879 non-null  object
 2   fnlwgt          48842 non-null  int64 
 3   education       48842 non-null  object
 4   education-num   48842 non-null  int64 
 5   marital-status  48842 non-null  object
 6   occupation      47876 non-null  object
 7   relationship    48842 non-null  object
 8   race            48842 non-null  object
 9   sex             48842 non-null  object
 10  capital-gain    48842 non-null  int64 
 11  capital-loss    48842 non-null  int64 
 12  hours-per-week  48842 non-null  int64 
 13  native-country  48568 non-null  object
 14  income          48842 non-null  object
dtypes: int64(6), object(9)
```

Figure 2: Dataset info

Using the pandas library in Python we were able to see dataset structure. The dataset has six numerical features age, fnlwgt, education-num, capital-gain, capital-loss, hours-per-week and eight categorical features workclass, education, marital-status, occupation, relationship, race, sex, native-country. The target variable to predict is the income column.

Feature Categories Analysis:

Demographic Features: Age, sex, race, and native-country gives personal identifying information. The Age ranges from 17 to 90 years. Race categories include White, Black, Asian-Pac-Islander, Amer-Indian-Eskimo, and Other.

Employment Features: The workclass feature includes Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, and Never-worked. Occupation feature has many different job categories including management and professional to service and labour roles. Hours-per-week is between 1 to 99 with median of approximately 40 hours.

Educational Features: education column Bachelors, Masters, Doctorate, HS-grad, etc. and also gives numerical encoding (education-num) ranging from 1 to 16.

Financial Features: Capital-gain and capital-loss show the investment income other than their work income. Most people have zero for these features. But when they do have some investments, these show high influence income levels. The fnlwgt variable is for the census sampling weights for population representation.

Data Quality Assessment: After checking for Missing values we saw that The categorical features workclass, occupation, and native-country had a '?' symbol meaning there are missing values. These and any other empty values were in 2799 workclass entries (5.73%), 2809 occupation entries (5.75%), and 857 native-country entries (1.75%).

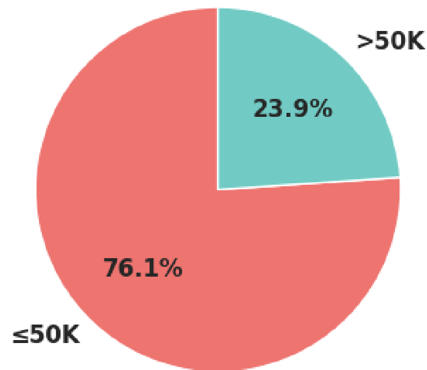


Figure 3: Income Distribution

After analysing the target variable we found there were some extra periods in the end of some values so we had to remove that. After cleaning we saw a big class imbalance.

37155 people (76.1%) earning ≤$50K and 11687 people (23.9%) earning >$50K. This might cause the models to have a bias towards the majority class and will be handled during the model training part.

2.3 Data Preparation

The data preparation for both Python and RapidMiner model implementations had systematic cleaning and data transformation. The preparation included handling missing data, encoding categorical columns and feature scaling and handling class imbalance.

Missing Value Treatment: There were many values having the '?' symbols which were replaced with 'Unknown' category for categorical features. This method allows us to keep the data and also be able to do encoding on. Numerical features with missing values were replaced using median values to keep distribution characteristics and avoid any outliers.

Target Variable Preparation: The Income labels also required cleaning to remove some formatting issues. Some entries had extra periods in the end that were removed using string operations. We then used LabelEncoder to map ≤50K to 0 and >50K to 1.

Categorical Encoding Strategies: Two different encoding techniques were used in python to improve performance for different types of algorithm.

1. **One-Hot Encoding** was used for linear algorithms. This method creates binary indicator variables for each category but drops one category to prevent multicollinearity.
2. **Label Encoding** was used for tree-based algorithms. This method is computationally efficient and keeps categorical relationships.

Feature Scaling Implementation: StandardScaler was used for numerical features as it is important for some types of models to work well.

Train-Test Split Methodology: The data was split into 80/20 ratio so that we have good amount of training data and also have enough data left for evaluating.

Class Imbalance Handling: We used SMOTE (Synthetic Minority Oversampling Technique) to fix the class imbalance on the training data after splitting to avoid data leakage. This method creates synthetic minority class instances using k-nearest neighbour method. After using SMOTE we had 50:50 distribution of target variable instead of the initial 76:24 we had. This helps in providing more data for the model to learn about both classes.

2.4 Modelling

2.4.1 RapidMiner AutoModel

The inbuilt functionality of RapidMiner's AutoModel feature gives a complete automated workflow for machine learning. It includes model training, hyperparameter tuning and evaluates performance by just dragging and dropping. The platform makes it easier for users to train models with almost no manual programming.

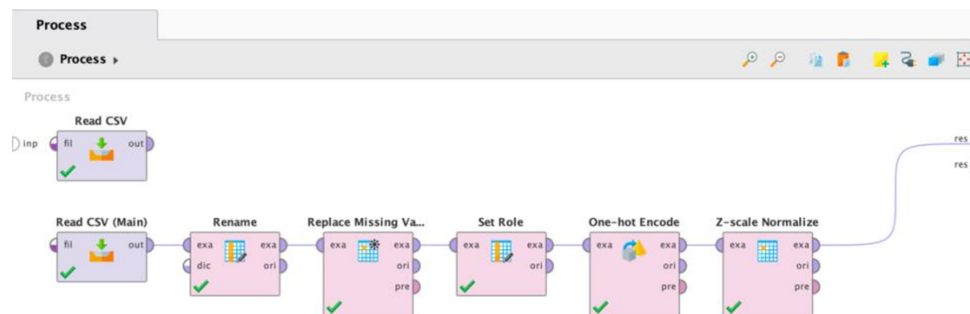


Figure 4: RapidMiner Workflow

The AutoModel workflow design is shown in Fig. 4. We can see how straightforward and easy it is to implement right from data loading to final preprocessing. The system handles all preprocessing steps and any transformations needed also automatically based on the dataset.

We used AutoModel to train five different classification algorithms - Logistic Regression, Decision Tree, Random Forest, XGBoost and Naive Bayes. AutoModel performed hyperparameter tuning, cross-validation and evaluated the performances of all of the models automatically. This helped in very fast model development and best results without the users needing to do any manual hyperparameter tuning which saved a lot of effort and time.

One of the main advantage of RapidMiner is how quickly it can get results for several algorithms which can then be further improved manually if needed. The results from AutoModel also provide lots of valuable information to understand how to improve their models without spending much development time.

RapidMiner Performance Results:

Model Name	Accuracy	Precision	Recall	F1-Score
XGBoost	83.5%	84.8%	95.3%	89.8%
Random Forest	82.4%	84.6%	93.8%	89%
Decision Tree	82.5%	83.7%	95.5%	89.2%
Logistic Regression	83.1%	84.7%	94.7%	89.4%
Naïve Bayes	75.9%	75.9%	100%	86.3%

From the results we can see we the recall is consistently high in four out of five models having more than 93% recall and Naïve Bayes having 100% recall. This shows that AutoModel prioritises in improving recall for high income individuals instead of precision. This is important for our business case as we want to focus on the high value individuals as missing the high earners will affect the business a lot more.

XGBoost was the best model overall with 83.5% accuracy and highest F1-score of 89.8% showing a very good balance in precision and recall. It was expected for the XGBoost model to perform the best because of the advanced ensemble logic it uses which has built in regularisation which is very good at handling feature interactions and also prevent any overfitting on the data.

2.4.2 Python

Python allows user to be able to fine tune every step in building the model. We had complete control in the development process from different preprocessing methods to configuring algorithms and evaluation methods using the scikit-learn and XGBoost library. This approach provided full model transparency and customisation opportunity.

Algorithm Configuration Specifications:

For Logistic Regression we slowly increased the maximum iterations (max_iter=1000) to get convergence. We kept the regularisation parameters at default to balance the model's complexity and generalisation.

With chose Gaussian Naive Bayes (GaussianNB()) which is more better for continuous features and also has very less computational costs. We kept the parameters as the default parameters to avoid any zero-probability scenarios.

Decision Tree used max_depth=15 to prevent any overfitting but also making sure to maintain enough model complexity to capture important patterns for income determination in the dataset.

Random Forest used the parameter, n_estimators=100 and also the max_depth=15 to make sure we have good enough performance while also making sure training time doesn't take too long and we could save computational resources.

XGBoost didn't need much configuration. Just using the log loss evaluation metric (eval_metric='logloss') was enough to make sure the model would work well for our business case and kept all other parameters the same.

Model Training and Evaluation Procedures: Each algorithm was trained systematically by using the fit method on our training data. Once training was done we used the predict method to get the predictions and predict_proba method for probability estimation on the testing data.

The performance on the models was done using the metrics available by scikit-learn. Accuracy was calculated using accuracy_score, Precision using precision_score, Recall using recall_score and F1-score using f1_score functions. These metrics helped in proper comparison of models.

Python Performance Results:

Model Name	Accuracy	Precision	Recall	F1-Score
XGBoost	86.9%	72.9%	72%	72.5%
Random Forest	84.9%	66.9%	72.9%	69.8%
Decision Tree	83.5%	64%	71%	67.3%
Logistic Regression	81%	57%	83.2%	67.7%
Naïve Bayes	53.6%	33.5%	95.2%	49.6%

The Python XGBoost model got the highest accuracy of 86.9% and was also the most balanced model when compared to others (F1-score of 72.5%) which makes it good for broad business use. All the tree based algorithms were better than the linear models which suggests that tree algorithms are better for complex non-linear relationships.

From the results it is also clear that our different preprocessing strategies helped improve the results for tree based algorithms. Label encoding is much better than one-hot encoding for categorical data which also saves computational resources and time. One-hot encoding with standard scaling made sure that the linear algorithms performed well.

Naïve Bayes had a very bad precision of 33.5% but the highest recall of 95.2% which shows that the model is good for identifying high earners but the very bad precision means it is not good for practical deployment.

3. Evaluation

We compared the results from both platforms for our evaluation phase. We compared all the standard metrics and compared them using some visualisation techniques and doing some error analysis. The analysis mainly focused on the differences in performances of automated (RapidMiner) and manual (Python) models.

Comprehensive Cross-Platform Performance Analysis:

Model Name	Python Accuracy	RapidMiner Accuracy	Difference
XGBoost	**86.9%**	83.5%	+ 3.3%
Random Forest	**84.9%**	82.4%	+ 2.5%
Decision Tree	**83.5%**	82.5%	+ 1.0%
Logistic Regression	81%	**83.1%**	- 2.1%
Naïve Bayes	53.6%	**75.9%**	- 22.3%

The Python models got better accuracy for XGBoost, Random Forest and Decision trees. XGBoost had the most improvement of +3.3 percentage, Random Forest had +2.5 percentage and Decision Tree had +1.0 percentage improvement. This suggests that our model specific preprocessing helped in slightly improving the accuracy but it didn't help much with precision and recall.

The tree-based ensemble methods (XGBoost, Random Forest) had better performance in all the metrics than the individual models. This suggests that the non-linear relationships in our dataset is better understood by the aggregation done by ensemble methods.

Algorithm-Specific Performance Insights:

XGBoost performed the best on both models getting the highest accuracy of 86.9% in Python and 83.5% in RapidMiner. This is because of how XGBoost properly handles complex feature interactions by doing sequential error correction and having built-in regularisation to prevent any overfitting.

The other Tree algorithms (Decision Tree, Random Forest) also showed good performances on all metrics on both platforms. The Python versions performed slightly better which could be due to the specific preprocessing strategies.

The linear algorithms show the most differences in performances in both tools. Both logistic regression and Naïve bayes performed better in RapidMiner with Naïve Bayes having the most improvement of 22.3%. This shows that the algorithms depend a lot on the hyperparameters and preprocessing strategies.

Considering all of the evaluation metrics accuracy, precision, recall and F1-score we found that the XGBoost in RapidMiner had the overall best performance which makes it the best for choice for real world usage. It has a very high recall of 95.3% and F1 score of 89.8% which makes it ideal for our business case even though it has slightly lower accuracy than the python version of XGBoost.

4. Model Deployment

After completing al the evaluation methods we decided the best model for our business case is the RapidMiner version of XGBoost model. It had one of the best recalls (95.3%) and the highest F1-score of 89.8% out of all the models in both tools. Although the python version of XGBoost had higher accuracy (3.3% more), the precision and recall were very less which meant we would miss a lot of the high earners which is not good for business.

Model Selection Justification: In income predictions it is important to be able to identify all the high earners and for that we had to prioritise models with high recall which RapidMiner's XGBoost gave us. The precision of 84.8% was also the highest in all models so we would also have very less false positives.

The F1-score also being the highest at 89.8% meant better balance in precision and recall. RapidMiner also provides more reliable and confident results as there is not much chance of human errors so it is good for deploying in production environment also.

Technical Deployment Architecture: RapidMiner has native deployment functionalities that help in integrating the models into business systems using standardised API interfaces. Since everything is automated the pipeline makes sure all preprocessing and data transformations are correct which makes sure all process we did remain the same and get expected results.

RapidMiner handles all encoding of categorical variables, missing value replacement and other feature engineering automatically which makes it less prone to any human errors making the implementation of the model easier for future users and have similar results. RapidMiner also has inbuilt functionality for tracking and monitoring the models performance so we can check if there are any issues as more data comes in the future.

Business Integration Applications: Our deployed model can be used for multiple different industries and help with different needs. Financial services can use it for advanced income verification in real time for checking of loan applications. The high recall of our model makes sure all high income people are a identified and they are not rejected.

Marketing departments can easily identify high earners so that they can create targeted marketing strategies for their premium products and make sure they cover all of them and none are left out. The high precision also makes sure that they are efficient in targeting their customers.

Government agencies can improve their eligibility screening process based on income using our model. The high recall again makes sure high earners are properly excluded from any programmes for low earners.

Operational Performance Characteristics: The easy setup and implementation of RapidMiner makes it suitable for real-time and large scale batch processing and also reduces the computational costs. RapidMiner was also built to ensure it supports high-volume architectures and provide very good flexibility.

Future Enhancement Opportunities: The automatic deployment structure helps in making more improvements to our model using better ensemble techniques and maybe adding more features for training. RapidMiner automatically retrains the models which allows to schedule regular model training so that it stays updated without manual involvement.

5. Conclusion

The project was able to successfully use the Adult Census Income dataset and create multiple models that could predict income of people using the CRISP-DM methodology. The systematic six stage cycle helped in getting our main goal of developing a model with high recall. Our models could classify if a person's income level is above or below $50,000 with high accuracy.

Implementing on two different platforms helped us understand the differences between automated and manual model development process. The python model development did have better accuracy for tree based algorithms because of the specific preprocessing steps, but did have lower recall and precisions. RapidMiner had slightly lower accuracies but had much better recall and precision because of the automated hyperparameter tuning.

From all the models we chose the RapidMiner version of XGBoost as it had the best performance with high accuracy (83.5%), recall (95.3%) and precision (84.8%). It also performed consistently well on both platforms making it suitable for real world deployment.

Key Findings: RapidMiner versions of the algorithms showed much better recall rates than manual implementation which shows how important it is to do hyperparameter tuning. Even though the manual implementation helped in getting better accuracies for tree algorithms, they had lower precision and recall.

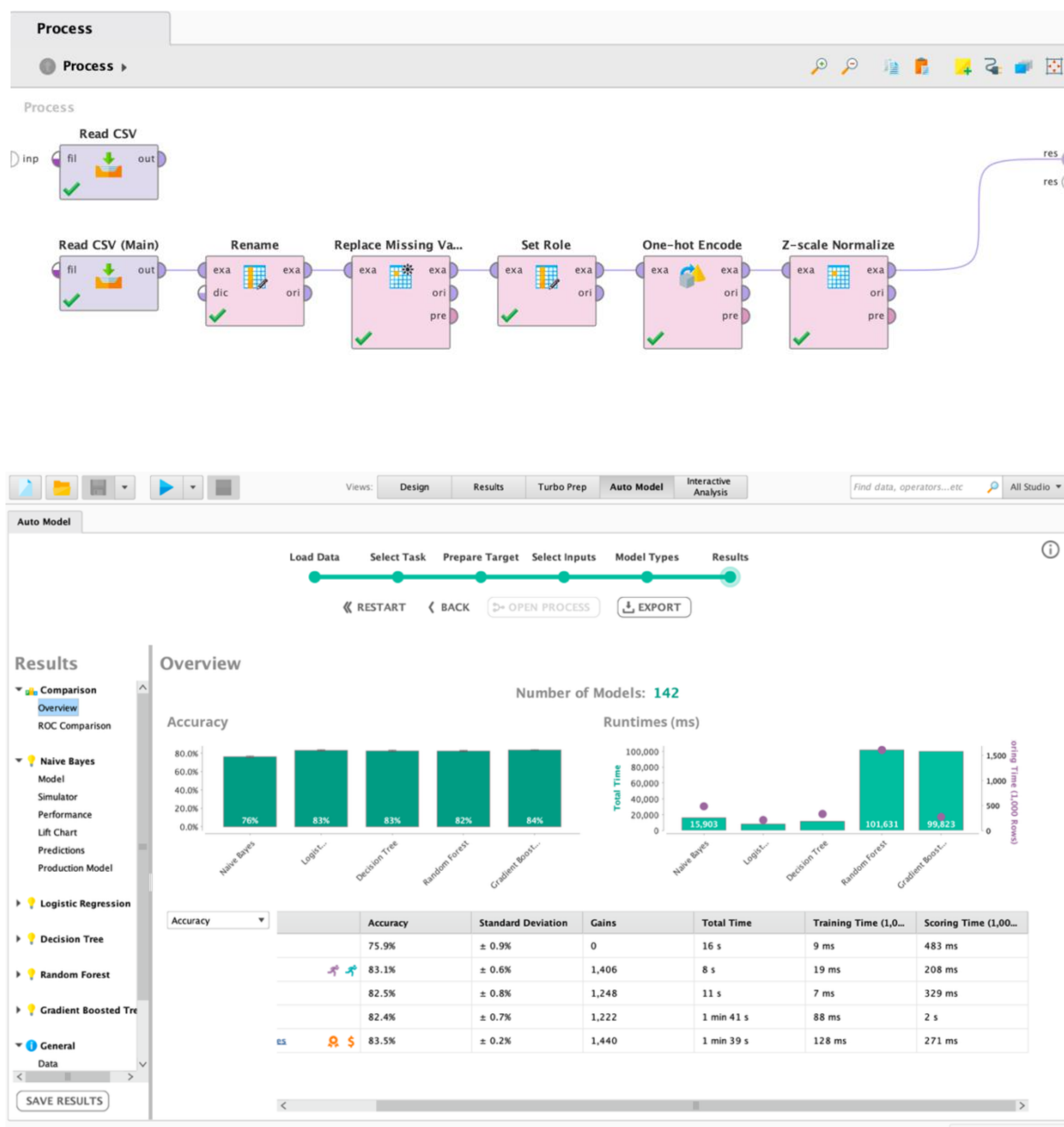
From a business context the models can be implemented by financial and government institutions and by marketing teams who need to assess the incomes of individuals. The selected RapidMiner XGBoost model provides very high recall of 95.3% and strong F1-score of 89.8% and accuracy of 83.5% making it very suitable in cases when it is very important to identify high income people.

References

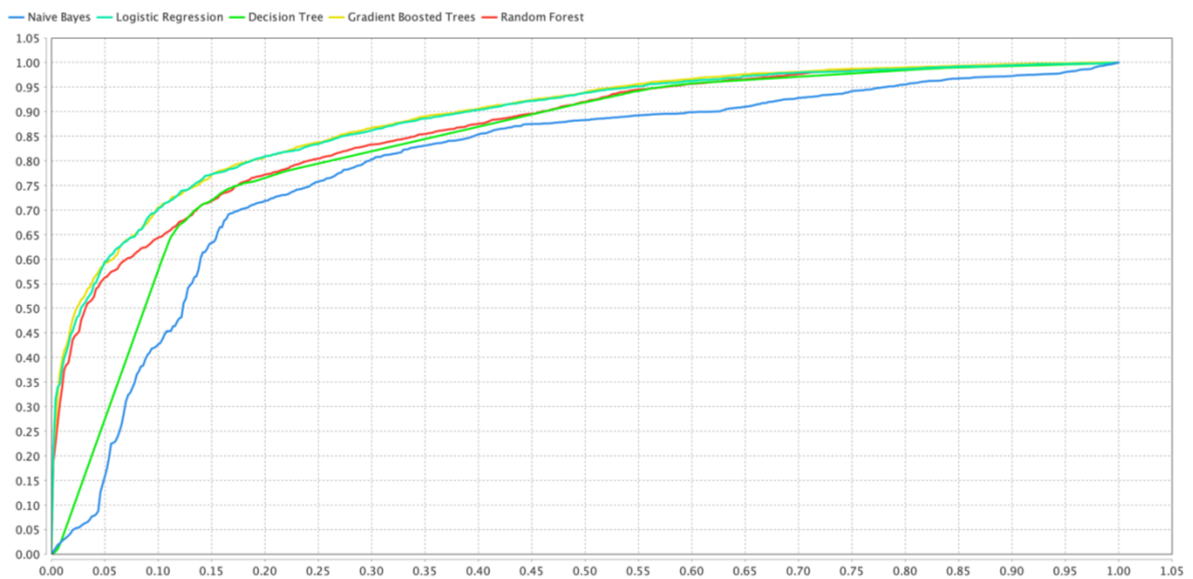
Becker, B. and Kohavi, R. (1996) Adult [Dataset]. UCI Machine Learning Repository. Available at: https://doi.org/10.24432/C5XW20

Wirth, R. and Hipp, J. (2000) CRISP-DM: Towards a Standard Process Model for Data Mining. *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining*, Manchester, pp. 29-39.

Appendix A – RapidMiner



ROC Comparison



Views: Design Results Turbo Prep **Auto Model** Interactive Analysis

Find data, operators...etc All Studio

Auto Model

Load Data · Select Task · Prepare Target · Select Inputs · Model Types · Results

RESTART BACK OPEN PROCESS EXPORT

Results

ROC Comparison

- **Naive Bayes**
  - Model
  - Simulator
  - Performance
  - Lift Chart
  - Predictions
  - Production Model
- **Logistic Regression**
- **Decision Tree**
- **Random Forest**
- **Gradient Boosted Tre**
- **General**
  - Data
  - Statistics
  - Weights by Correlation

SAVE RESULTS

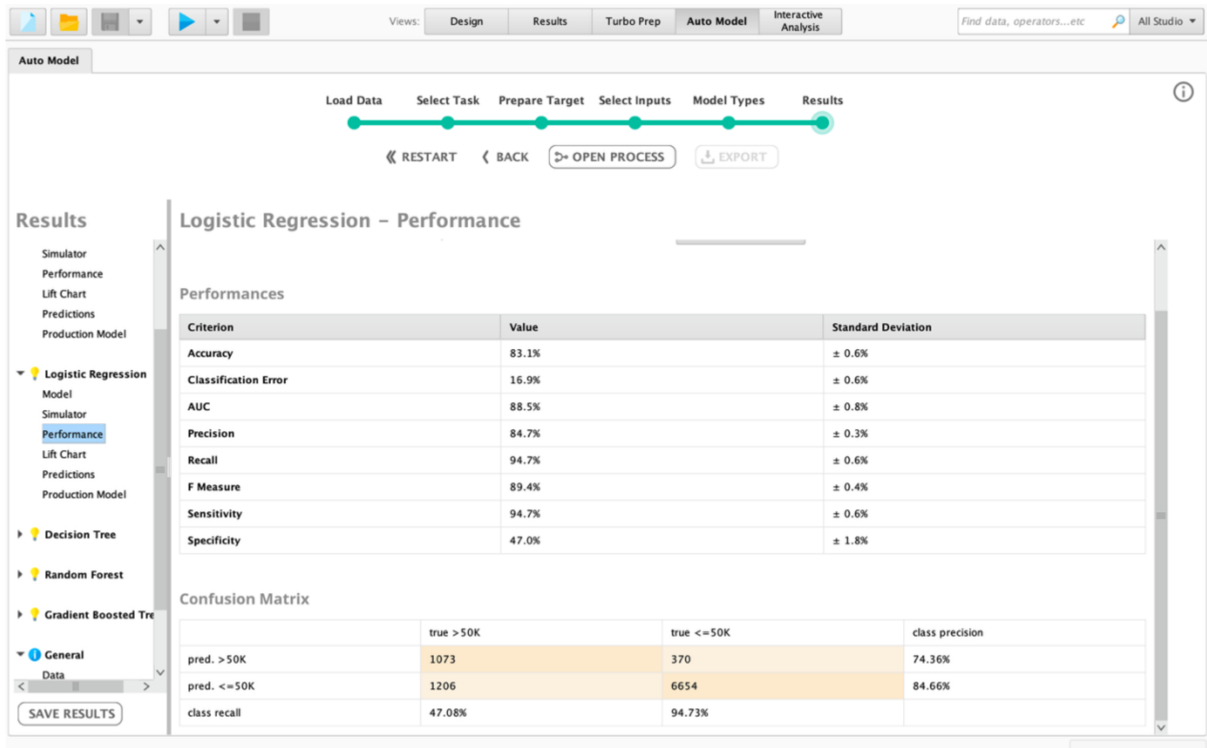
Naive Bayes – Performance

Performances

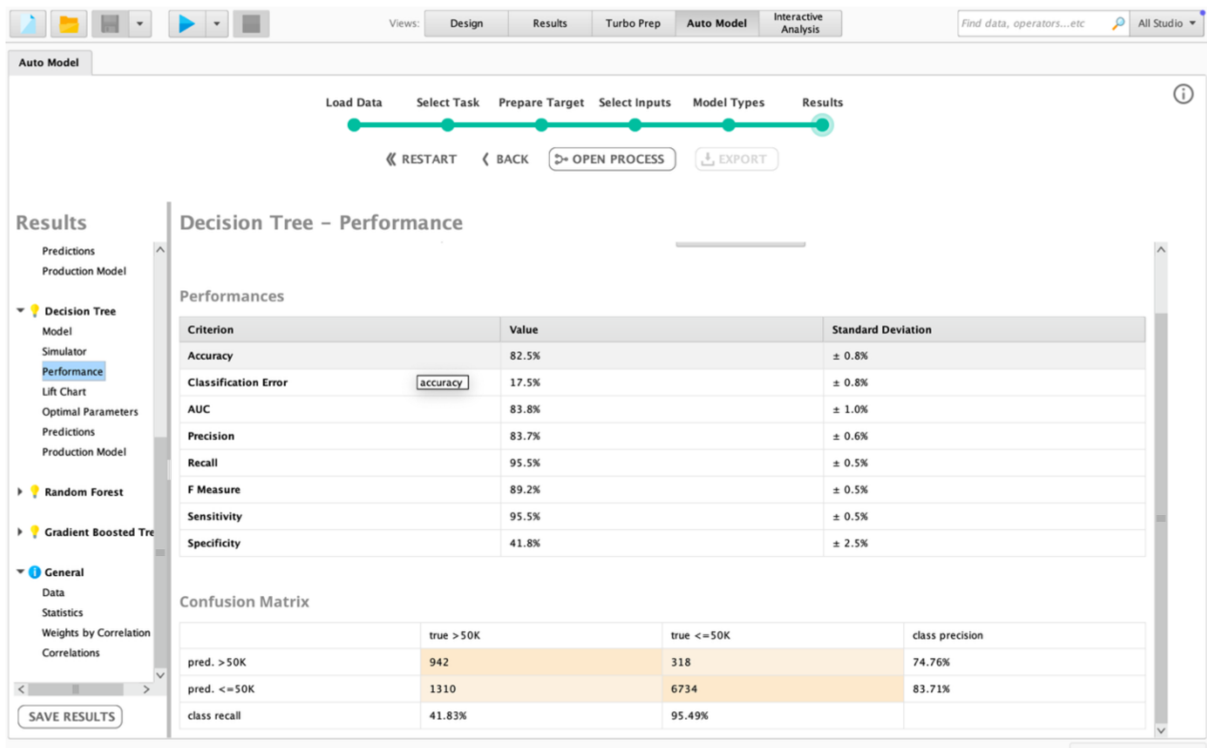
Criterion	Value	Standard Deviation
Accuracy	75.9%	± 0.9%
Classification Error	24.1%	± 0.9%
AUC	79.4%	± 1.4%
Precision	75.9%	± 0.9%
Recall	100.0%	± 0.0%
F Measure	86.3%	± 0.5%
Sensitivity	100.0%	± 0.0%
Specificity	0.0%	± 0.0%

Confusion Matrix

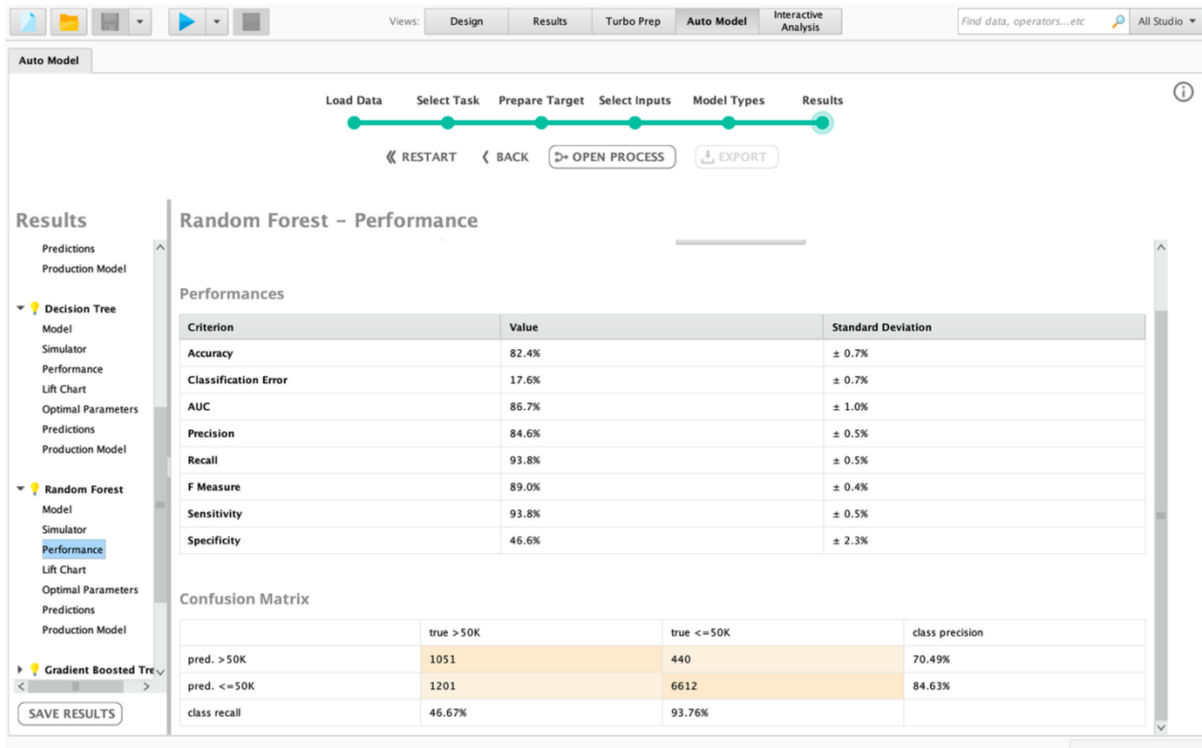
	true >50K	true <=50K	class precision
pred. >50K	0	0	0.00%
pred. <=50K	2241	7062	75.91%
class recall	0.00%	100.00%	

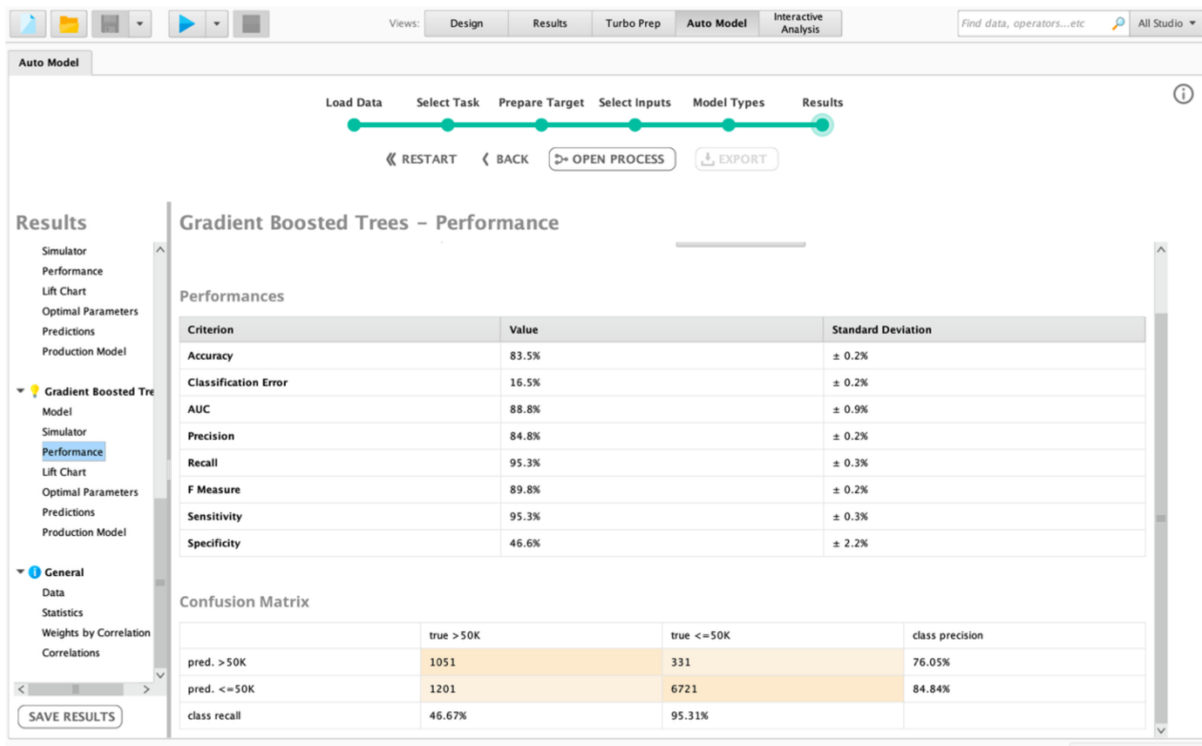


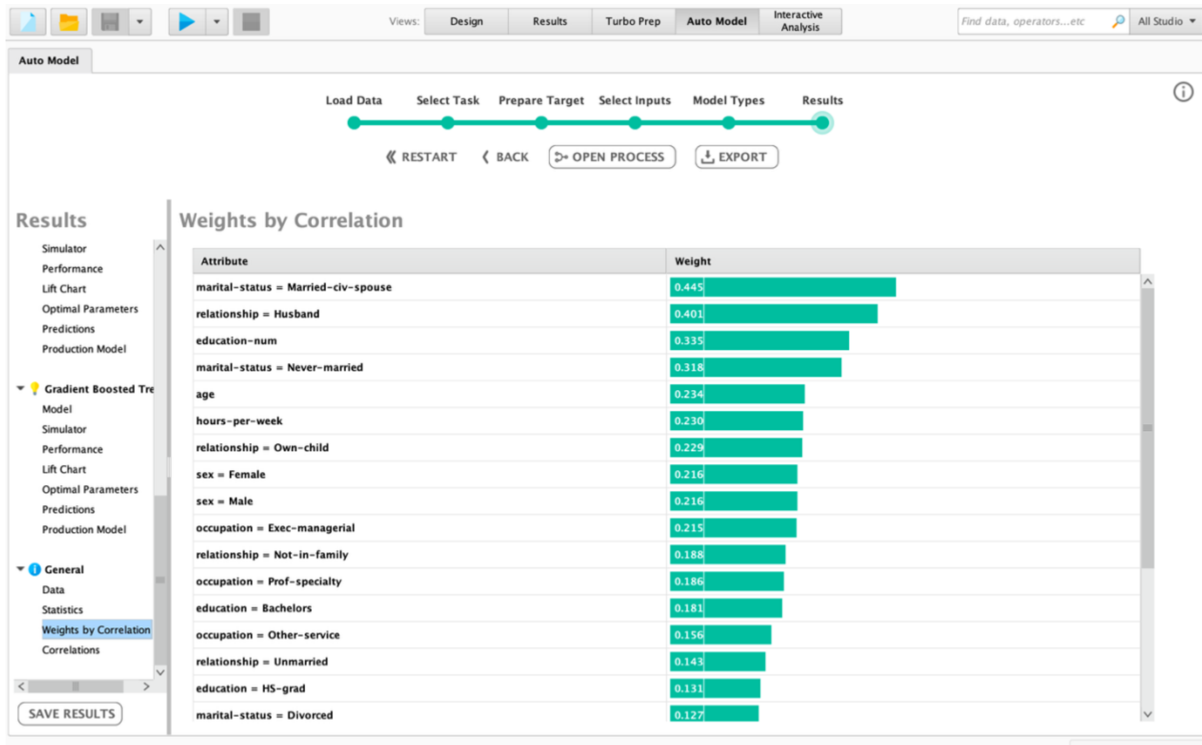
Views:
Design
Results
Turbo Prep
Auto Model
Interactive Analysis
Find data, operators...etc
All Studio
Auto Model
Load Data
Select Task
Prepare Target
Select Inputs
Model Types
Results
RESTART
BACK
OPEN PROCESS
EXPORT
Results
Simulator
Performance
Lift Chart
Predictions
Production Model
Logistic Regression
Model
Simulator
Performance
Lift Chart
Predictions
Production Model
Decision Tree
Random Forest
Gradient Boosted Tre
General
Data
SAVE RESULTS
Logistic Regression – Performance
Performances
Criterion | Value | Standard Deviation
Accuracy | 83.1% | ± 0.6%
Classification Error | 16.9% | ± 0.6%
AUC | 88.5% | ± 0.8%
Precision | 84.7% | ± 0.3%
Recall | 94.7% | ± 0.6%
F Measure | 89.4% | ± 0.4%
Sensitivity | 94.7% | ± 0.6%
Specificity | 47.0% | ± 1.8%
Confusion Matrix
| true >50K | true <=50K | class precision
pred. >50K | 1073 | 370 | 74.36%
pred. <=50K | 1206 | 6654 | 84.66%
class recall | 47.08% | 94.73% |



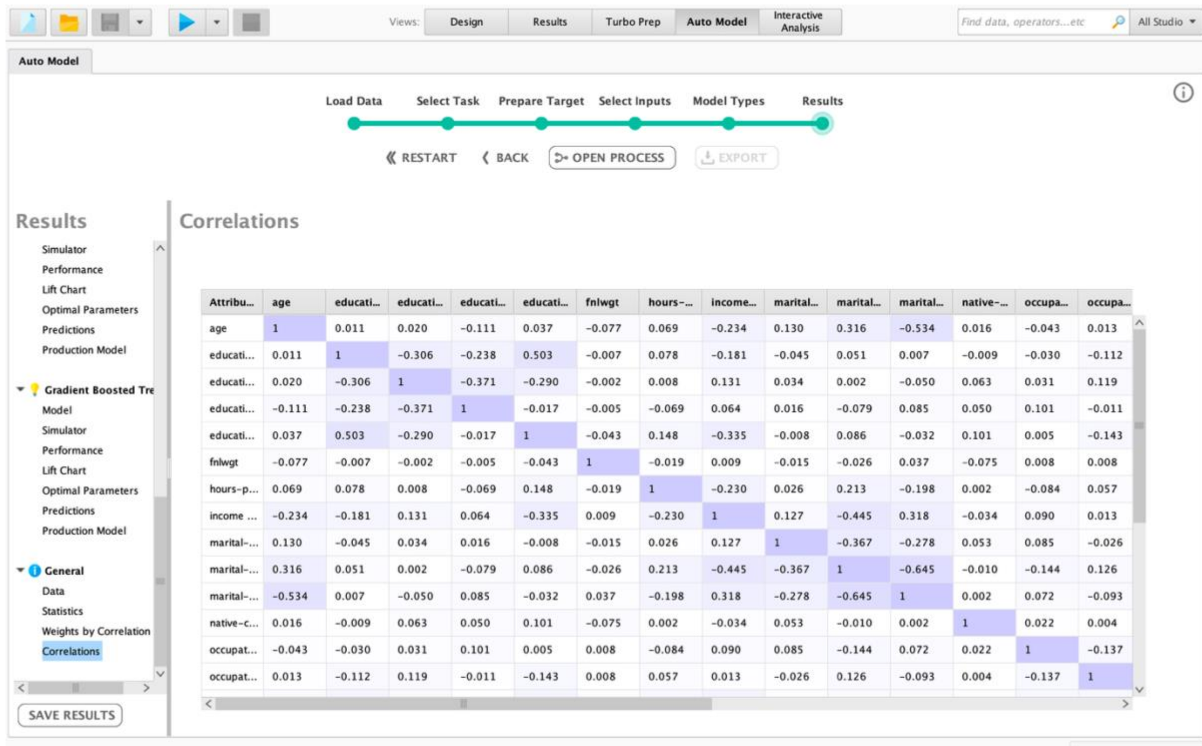
Views:
Design
Results
Turbo Prep
Auto Model
Interactive Analysis
Find data, operators...etc
All Studio
Auto Model
Load Data
Select Task
Prepare Target
Select Inputs
Model Types
Results
RESTART
BACK
OPEN PROCESS
EXPORT
Results
Predictions
Production Model
Decision Tree
Model
Simulator
Performance
Lift Chart
Optimal Parameters
Predictions
Production Model
Random Forest
Gradient Boosted Tre
General
Data
Statistics
Weights by Correlation
Correlations
SAVE RESULTS
Decision Tree – Performance
Performances
Criterion | Value | Standard Deviation
Accuracy | 82.5% | ± 0.8%
Classification Error | 17.5% | ± 0.8%
accuracy
AUC | 83.8% | ± 1.0%
Precision | 83.7% | ± 0.6%
Recall | 95.5% | ± 0.5%
F Measure | 89.2% | ± 0.5%
Sensitivity | 95.5% | ± 0.5%
Specificity | 41.8% | ± 2.5%
Confusion Matrix
| true >50K | true <=50K | class precision
pred. >50K | 942 | 318 | 74.76%
pred. <=50K | 1310 | 6734 | 83.71%
class recall | 41.83% | 95.49% |


Views:
Design
Results
Turbo Prep
Auto Model
Interactive Analysis
Find data, operators...etc
All Studio
Auto Model
Load Data
Select Task
Prepare Target
Select Inputs
Model Types
Results
RESTART
BACK
OPEN PROCESS
EXPORT
Results
Predictions
Production Model
Decision Tree
Model
Simulator
Performance
Lift Chart
Optimal Parameters
Predictions
Production Model
Random Forest
Model
Simulator
Performance
Lift Chart
Optimal Parameters
Predictions
Production Model
Gradient Boosted Tre
SAVE RESULTS
Random Forest – Performance
Performances
Criterion | Value | Standard Deviation
Accuracy | 82.4% | ± 0.7%
Classification Error | 17.6% | ± 0.7%
AUC | 86.7% | ± 1.0%
Precision | 84.6% | ± 0.5%
Recall | 93.8% | ± 0.5%
F Measure | 89.0% | ± 0.4%
Sensitivity | 93.8% | ± 0.5%
Specificity | 46.6% | ± 2.3%
Confusion Matrix
true >50K | true <=50K | class precision
pred. >50K | 1051 | 440 | 70.49%
pred. <=50K | 1201 | 6612 | 84.63%
class recall | 46.67% | 93.76%


Views:
Design
Results
Turbo Prep
Auto Model
Interactive Analysis
Find data, operators...etc
All Studio
Auto Model
Load Data
Select Task
Prepare Target
Select Inputs
Model Types
Results
RESTART
BACK
OPEN PROCESS
EXPORT
Results
Simulator
Performance
Lift Chart
Optimal Parameters
Predictions
Production Model
Gradient Boosted Tre
Model
Simulator
Performance
Lift Chart
Optimal Parameters
Predictions
Production Model
General
Data
Statistics
Weights by Correlation
Correlations
SAVE RESULTS
Gradient Boosted Trees – Performance
Performances
Criterion | Value | Standard Deviation
Accuracy | 83.5% | ± 0.2%
Classification Error | 16.5% | ± 0.2%
AUC | 88.8% | ± 0.9%
Precision | 84.8% | ± 0.2%
Recall | 95.3% | ± 0.3%
F Measure | 89.8% | ± 0.2%
Sensitivity | 95.3% | ± 0.3%
Specificity | 46.6% | ± 2.2%
Confusion Matrix
true >50K | true <=50K | class precision
pred. >50K | 1051 | 331 | 76.05%
pred. <=50K | 1201 | 6721 | 84.84%
class recall | 46.67% | 95.31%



Views: Design Results Turbo Prep Auto Model Interactive Analysis
Find data, operators...etc
All Studio
Auto Model
Load Data
Select Task
Prepare Target
Select Inputs
Model Types
Results
RESTART
BACK
OPEN PROCESS
EXPORT
Results
Simulator
Performance
Lift Chart
Optimal Parameters
Predictions
Production Model
Gradient Boosted Tre
Model
Simulator
Performance
Lift Chart
Optimal Parameters
Predictions
Production Model
General
Data
Statistics
Weights by Correlation
Correlations
SAVE RESULTS
Weights by Correlation
Attribute
Weight
marital-status = Married-civ-spouse 0.445
relationship = Husband 0.401
education-num 0.335
marital-status = Never-married 0.318
age 0.234
hours-per-week 0.230
relationship = Own-child 0.229
sex = Female 0.216
sex = Male 0.216
occupation = Exec-managerial 0.215
relationship = Not-in-family 0.188
occupation = Prof-specialty 0.186
education = Bachelors 0.181
occupation = Other-service 0.156
relationship = Unmarried 0.143
education = HS-grad 0.131
marital-status = Divorced 0.127

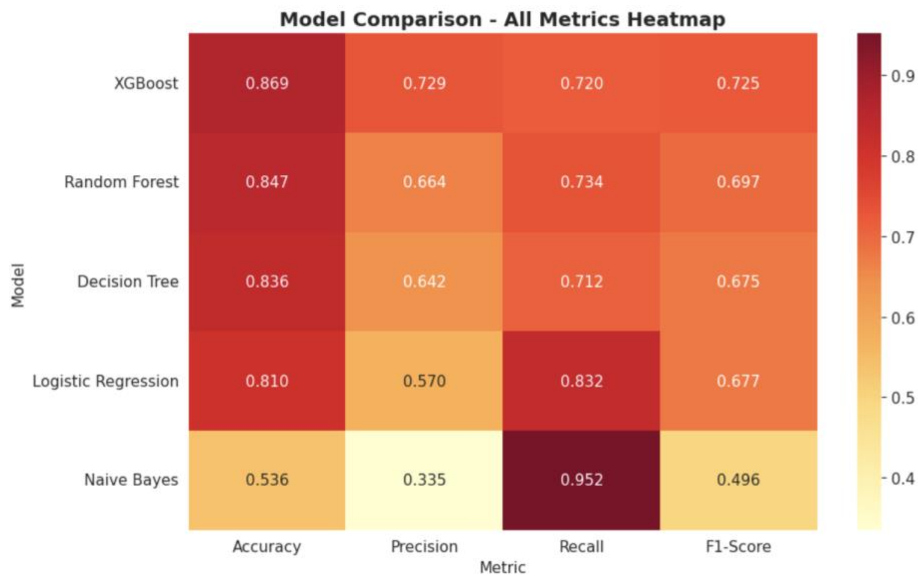


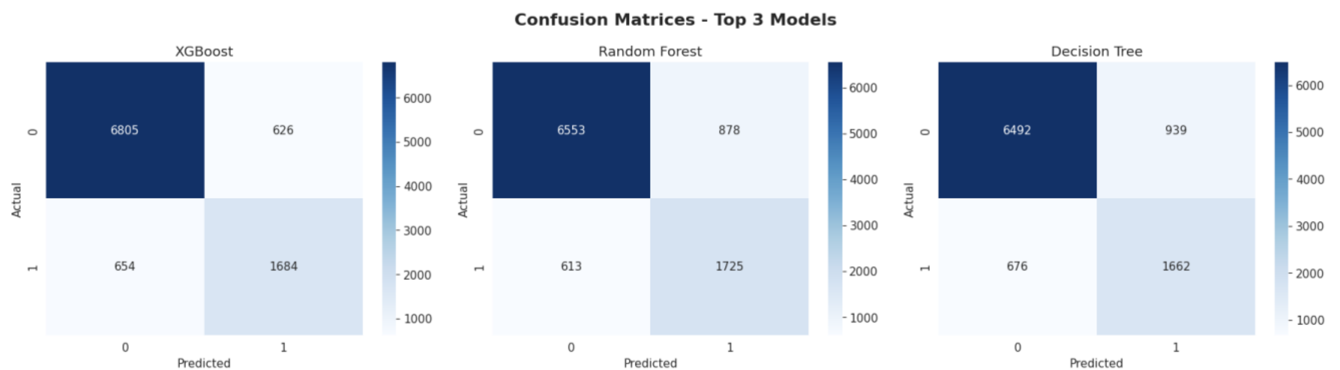
Views: Design Results Turbo Prep Auto Model Interactive Analysis
Find data, operators...etc
All Studio
Auto Model
Load Data
Select Task
Prepare Target
Select Inputs
Model Types
Results
RESTART
BACK
OPEN PROCESS
EXPORT
Results
Simulator
Performance
Lift Chart
Optimal Parameters
Predictions
Production Model
Gradient Boosted Tre
Model
Simulator
Performance
Lift Chart
Optimal Parameters
Predictions
Production Model
General
Data
Statistics
Weights by Correlation
Correlations
SAVE RESULTS
Correlations
Attribu... age educati... educati... educati... educati... fnlwgt hours-... income... marital... marital... marital... native-... occupa... occupa...
age 1 0.011 0.020 -0.111 0.037 -0.077 0.069 -0.234 0.130 0.316 -0.534 0.016 -0.043 0.013
educati... 0.011 1 -0.306 -0.238 0.503 -0.007 0.078 -0.181 -0.045 0.051 0.007 -0.009 -0.030 -0.112
educati... 0.020 -0.306 1 -0.371 -0.290 -0.002 0.008 0.131 0.034 0.002 -0.050 0.063 0.031 0.119
educati... -0.111 -0.238 -0.371 1 -0.017 -0.005 -0.069 0.064 0.016 -0.079 0.085 0.050 0.101 -0.011
educati... 0.037 0.503 -0.290 -0.017 1 -0.043 0.148 -0.335 -0.008 0.086 -0.032 0.101 0.005 -0.143
fnlwgt -0.077 -0.007 -0.002 -0.005 -0.043 1 -0.019 0.009 -0.015 -0.026 0.037 -0.075 0.008 0.008
hours-p... 0.069 0.078 0.008 -0.069 0.148 -0.019 1 -0.230 0.026 0.213 -0.198 0.002 -0.084 0.057
income ... -0.234 -0.181 0.131 0.064 -0.335 0.009 -0.230 1 0.127 -0.445 0.318 -0.034 0.090 0.013
marital-... 0.130 -0.045 0.034 0.016 -0.008 -0.015 0.026 0.127 1 -0.367 -0.278 0.053 0.085 -0.026
marital-... 0.316 0.051 0.002 -0.079 0.086 -0.026 0.213 -0.445 -0.367 1 -0.645 -0.010 -0.144 0.126
marital-... -0.534 0.007 -0.050 0.085 -0.032 0.037 -0.198 0.318 -0.278 -0.645 1 0.002 0.072 -0.093
native-c... 0.016 -0.009 0.063 0.050 0.101 -0.075 0.002 -0.034 0.053 -0.010 0.002 1 0.022 0.004
occupat... -0.043 -0.030 0.031 0.101 0.005 0.008 -0.084 0.090 0.085 -0.144 0.072 0.022 1 -0.137
occupat... 0.013 -0.112 0.119 -0.011 -0.143 0.008 0.057 0.013 -0.026 0.126 -0.093 0.004 -0.137 1

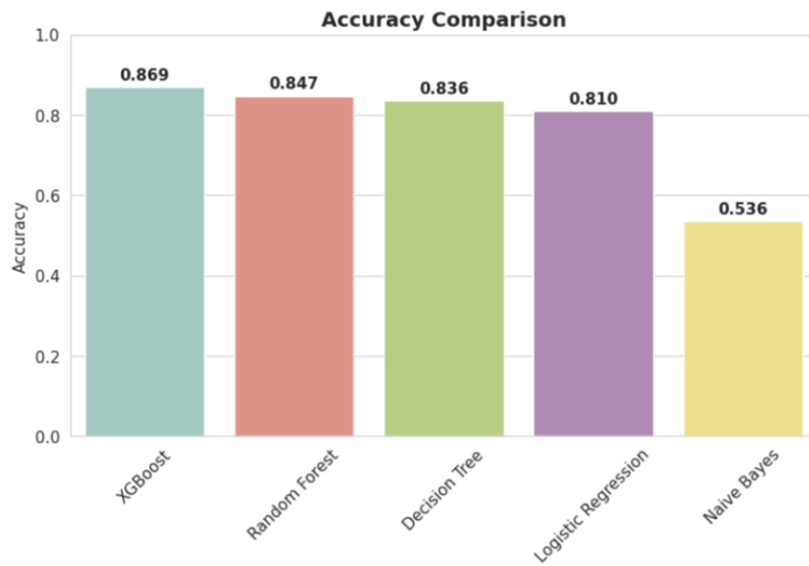
Appendix B – Python

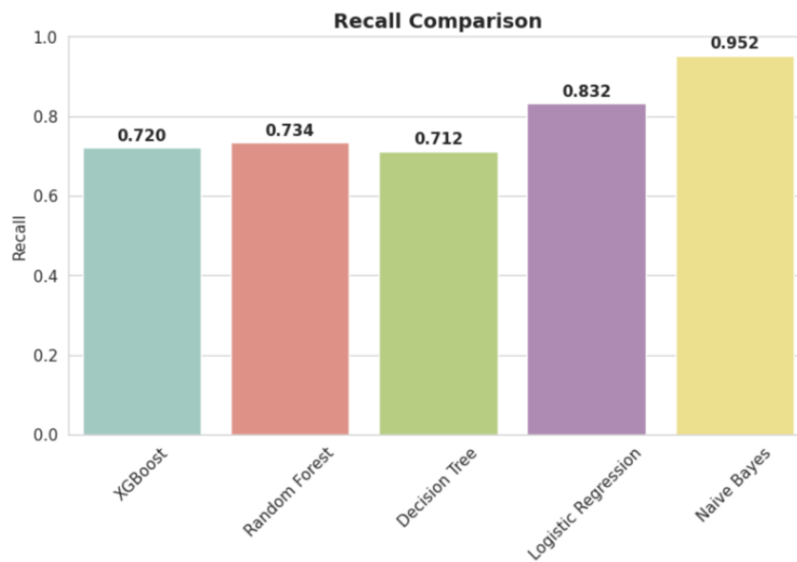
```
Missing values per column:
age                0
workclass        963
fnlwgt             0
education          0
education-num      0
marital-status     0
occupation       966
relationship       0
race               0
sex                0
capital-gain       0
capital-loss       0
hours-per-week     0
native-country   274
income             0
dtype: int64

'?' symbols per column:
workclass: 1836 (3.8%)
occupation: 1843 (3.8%)
native-country: 583 (1.2%)
```

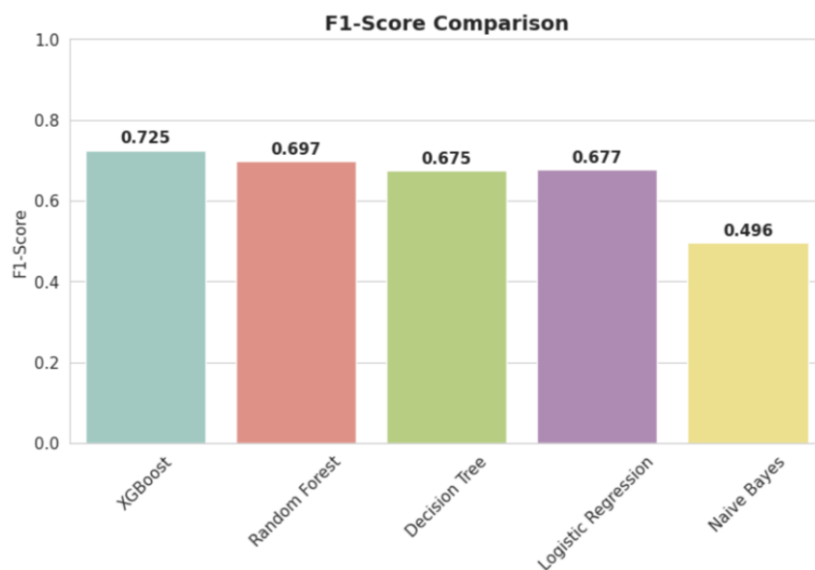
```
 income
<=50K      24720
<=50K.     12435
>50K        7841
>50K.       3846
Name: count, dtype: int64

After cleaning target variable
income
<=50K     37155
>50K      11687
Name: count, dtype: int64
```

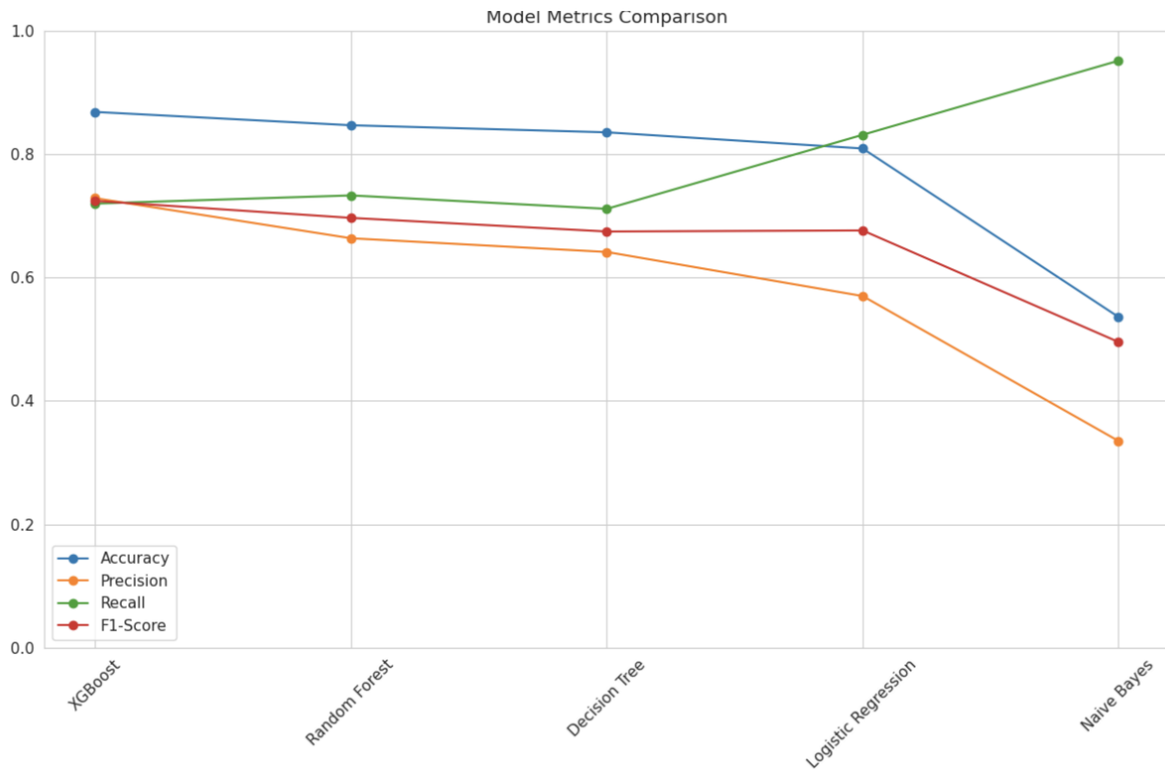


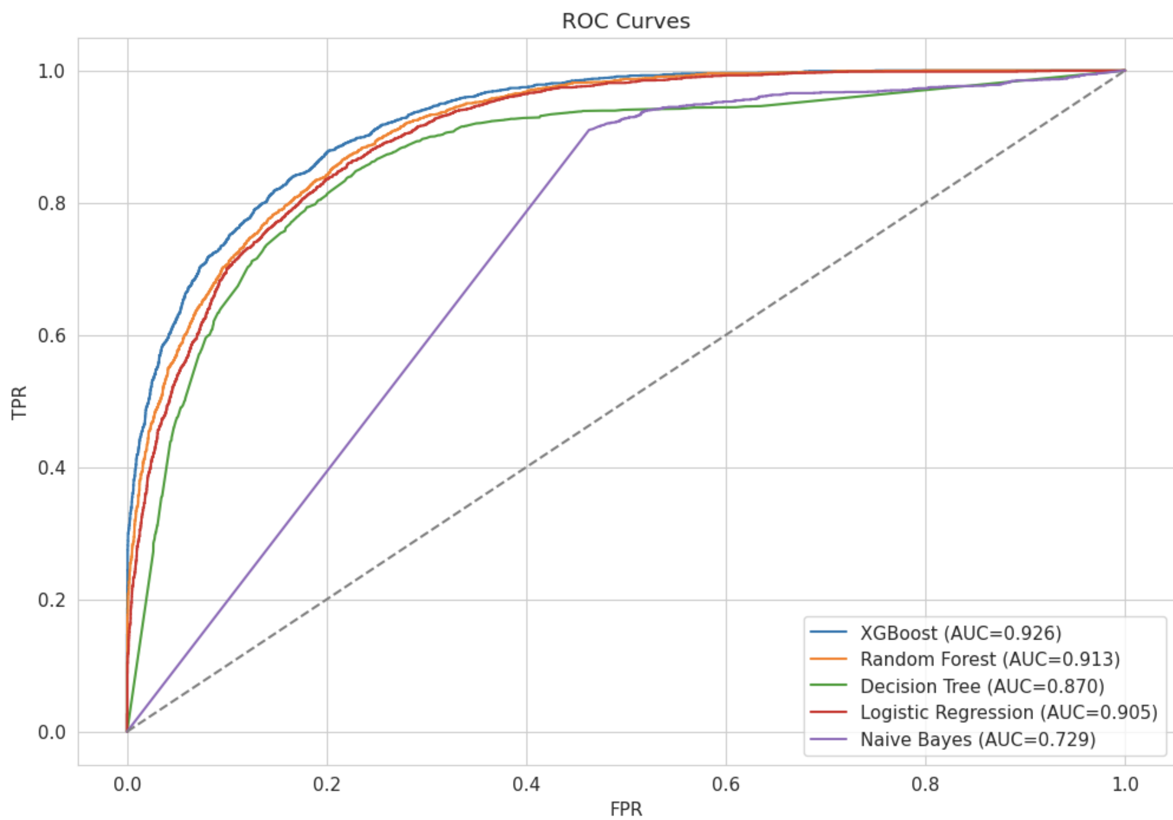



Accuracy Comparison
1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
0.869
0.847
0.836
0.810
0.536
XGBoost
Random Forest
Decision Tree
Logistic Regression
Naive Bayes


Recall Comparison
1.0
0.8
0.6
0.4
0.2
0.0
Recall
0.720
0.734
0.712
0.832
0.952
XGBoost
Random Forest
Decision Tree
Logistic Regression
Naive Bayes


F1-Score Comparison
1.0
0.8
0.6
0.4
0.2
0.0
F1-Score
0.725
0.697
0.675
0.677
0.496
XGBoost
Random Forest
Decision Tree
Logistic Regression
Naive Bayes


Model Metrics Comparison
1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
Precision
Recall
F1-Score
XGBoost
Random Forest
Decision Tree
Logistic Regression
Naive Bayes


ROC Curves
TPR
FPR
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
XGBoost (AUC=0.926)
Random Forest (AUC=0.913)
Decision Tree (AUC=0.870)
Logistic Regression (AUC=0.905)
Naive Bayes (AUC=0.729)