# BOT PREDICTION IN THE AUCTION MARKET

Project Report

ANUSHA MEDIBOINA (A0262847U)
MUTHUKUMARAN S/O SAMIAYYAN (A0027474A)
NAVEEN MATHEW VERGHESE (A0262734A)
PRASANNA GOVINDARAJAN (A0262732H)

NAVEEN MATHEW VERGHESE

# Contents

# Introduction

## Proposal / Objective

The objective of this project is to understand the problems posed by malicious bots on the auction market leading to impact on business profitability and customer dissatisfaction. We propose to design a model with the help of data analysis and machine learning to identify and eliminate such bots with the aid of Bot Prediction Lifecycle. Finally, we intend to show the business application using Expected Value framework to test for deployment capability.

## Background

The pandemic brought along with it, an urgent need for businesses to digitalize. Accordingly, the change in business model to online mode has created an environment for malicious bots to profit from digital platforms such as websites, mobile applications, and APIs and these bots cost businesses millions of dollars every year.

In 2020, about 2/3rd of businesses detected website attacks and there have also been innumerable instances of attacks on mobile apps and APIs. These businesses generally operate at very thin margins and bots cost them ~3 – 4% of their revenue. Businesses understand the challenges posed by Bots and the consequent effect on profitability and customer satisfaction. Further, ~4 – 18% of the customers are less likely to return to the platform.

It may be noted that many bot attacks originate from the same place of the affected business. Not all bot activities are illegal, for ex., while sniper and scalper bots may affect reputations and be against the terms and conditions of the business, there may not be an actual crime.

As per a survey by NETACEA[1] on 440 businesses in the UK and USA across multiple industries, bots resulted in profitability impact in the range of 3.2% - 3.9% of their revenue. Further, it took the businesses between 3.05 – 3.44 months to even identify the attack.

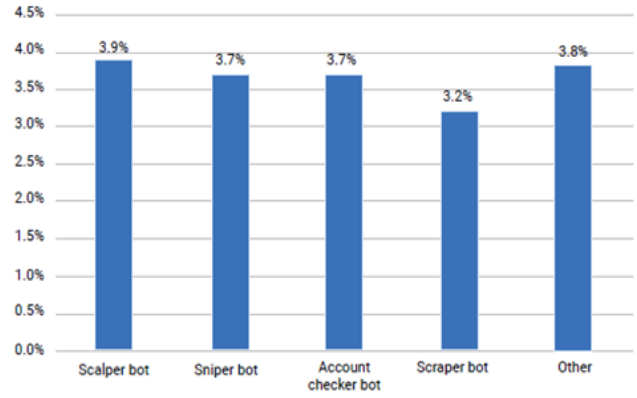



*Chart 1: Impact on Profitability*

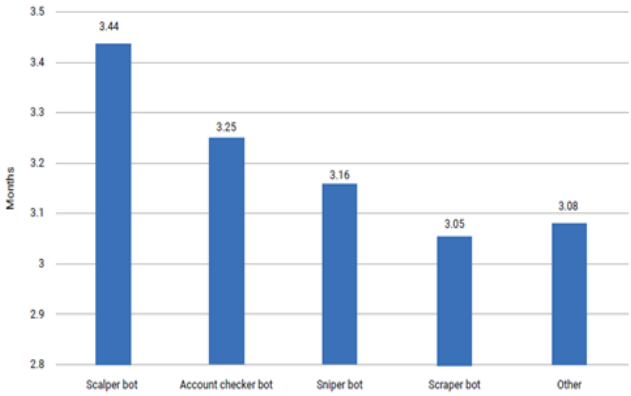



*Chart 2: Avg. time to recognize the attack*

A brief on each of the major categories of bots are given below[2]:

- **Sniper Bots:** works by automatically placing a bid on an auction item at the last possible second. The mechanism by which it operates is once the Bot operator enters a maximum bid amount (threshold) for an auction item, the bot will incrementally bid higher until the maximum threshold is reached after which it will stop. The process happens very fast and consequently, is difficult for other bidders to keep up. One key challenge with sniper bots is that it can be used to drive up the prices of auctions by placing multiple higher bids

  Some of the most popular web-based sniper bots are:
  - Auction Sniper: works with a variety of websites including eBay and Amazon
  - Trait Sniper: aims to purchase NFTs at the lowest possible price

- **Scalper Bots:** works by buying up large quantities of popular items and then resell at a higher price. To be noted that scalpers often prevent other customers from being able to purchase the desired item. Graphics cards and games console are examples of items snapped by bots and resold at higher prices.

- **Scraper Bots:** used to collect large amounts of data from websites for use elsewhere

While bots cause inconvenience, it also causes harm to businesses in the form of draining website resources, slowing them and harming business reputation by acquiring items faster than genuine customers can purchase them. A

customer who feels continuously let down by being unable to acquire the items in auction becomes an unhappy customer and is prone to leaving the online platform. As indicated in a report[3] (refer appendix), likelihood that bidders who were sniped do not return to bid on eBay was about 3.5% higher than those who lose without being sniped.
To be noted that a genuine customer will not be able to identify bot attacks, however, they may feel the impact of bots in the form of slow websites and failed auction bids. These customers tend to shift their businesses to other websites / auction houses.
The above cited issues necessitate an urgent need to design and deploy solutions intended to identify and eliminate bot attacks. Care should be taken to avoid incorrect identification of humans as bots as it could lead to loss of customer base and reputation. Similarly, unhappy customers who are unable to win bids tend to shift their businesses and can also cause reputation risk through negative publicity. Businesses need to allocate IT security budget to assess, identify and resolve issues arising from Bots on a regular basis.

# Bot Prediction Lifecycle

## General workflow of business analytics approach applied to current study

Based on the background and objective of the use case, the following Bot Prediction Lifecycle is proposed to help solve the problem statement. The steps of this lifecycle are aligned with the business analytics lifecycle.

1. **Business Understanding:** Understanding the impact of bot sniping on auctions and the need to eliminate them. The desired outcome is to retain the genuine customers from leaving the platform due to dissatisfaction (as they are unable to win bids)
2. **Data Acquisition:** Collecting data on bidder (describing whether it is a bot or a human) and the bids of various auctions (describing how, what and when of a bid)
3. **Data Preparation:** Merging different datasets, engineering the features to create more useful features for better predictability. Plotting the density functions to understand the importance of different features.
4. **Modelling:** Designing the cost-benefit matrix based on business understanding and the profitability of the prediction. Implementing the machine learning models for training and testing the data.
5. **Evaluation:** Comparing the performance metrics like accuracy, precision, recall, f1 and Expected value of all models. Identifying the best performing model.
6. **Deployment:** Comparing against the scenario where there is no model to predict the bots, using Expected value framework to check whether the model is ready for deployment.

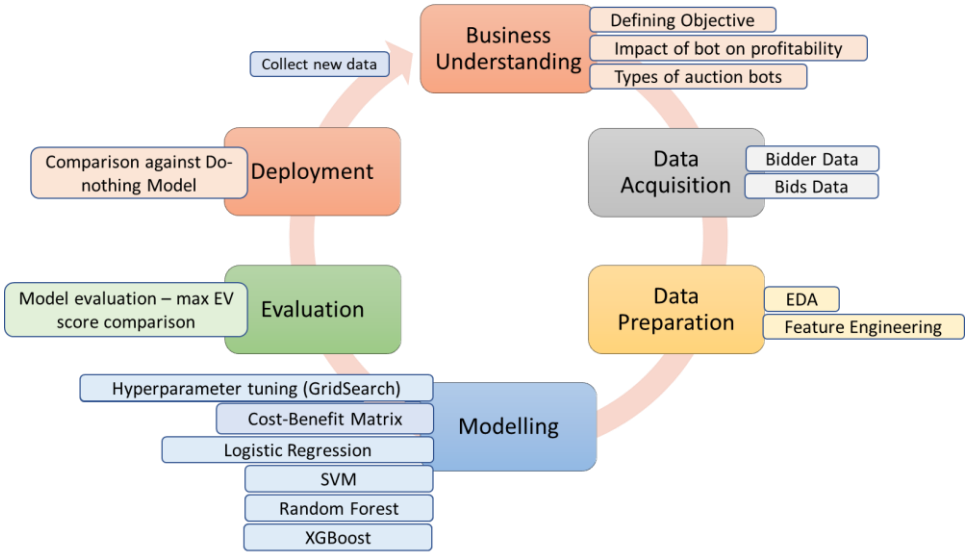



*Figure 1: Bot Prediction Lifecycle*

However, as shown in the figure above, this lifecycle is an iterative process. As more data is collected and business understanding is updated based on business metrics such as customer base retention and revenue loss due to bots, the model will be trained and updated and deployed in a periodic manner to identify bots from human bidders.

## Dataset

Two datasets from Kaggle(https://www.kaggle.com/code/chongzhenjie/human-or-robot) was used for the analysis and solution proposal of this business problem. One dataset contained the bidder information of the online auction platform consisting of 2013 unique bidders. The second dataset contained the information of the bids bid by the bidders for every auction present in the online auction platform (consisting of a total of 7647475 bids in total). The bidder information dataset consisted mainly of bidder id and the outcome of the bidders (target variable – if they are bid or not). The dataset is highly imbalanced as the number of bots is only 103 compared to 1910 humans in the bidder id dataset. The number of bots is only 5% of total bidder information dataset.

The bids dataset consisted mainly of the information on the bidder id making the bid, the auction id, the merchandise category that was bid and the device, country, IP address and URL from which the bid was made. The time and URL variable data were obfuscated for privacy purposes but the sequential nature of the time variable is preserved. A summary of the two datasets is shown in **Error! Reference source not found.** and Table 2 below.

| Column Name | Description |
|---|---|
| **Bidder_id** | Unique ID of bidder |
| **payment_account** | Payment account of bidder (obfuscated to protect privacy) |
| **address** | Mailing address of a bidder (obfuscated to protect privacy) |
| **Outcome** | Label of a bidder indicating whether it is a robot. Value 1.0 indicated robot. Value 0.0 indicated human – Target variable |

*Table 1: Dataset of bidder information – 2013 bidders*

| Column Name | Description |
|---|---|
| **bid_id** | Unique ID of bid |
| **bidder_id** | Unique ID of bidder |
| **auction** | Unique ID of an auction |
| **merchandise** | Category of auction site campaign |
| **device** | Phone model of visitor |
| **time** | Time that the bid is made (transformed to protect privacy) |
| **country** | Country that the IP belongs to |
| **ip** | IP address of bidder (obfuscated to protect privacy) |
| **url** | URL where the bidder was referred from (obfuscated to protect privacy) |

*Table 2 Dataset of bids information on the online platform – 7647475 bids*

# Feature Engineering

Before performing feature engineering on the dataset to create more useful features, the two datasets were first transformed and merged to 1 dataset. The bids information dataset was first grouped by the unique bidder ids and an aggregation of the non-unique counts of the rest of the features was performed. This data set was merged with the outcome feature (target variable) from the bidder information dataset. The payment_account and address features were dropped from this dataset as it was considered not useful for modelling as the data was obfuscated. Therefore, for each bidder ID, there were 8 features and 1 target variable in the transformed dataset.

More features were then created using the bids information dataset and merged with this transformed dataset. Even though the time feature was obfuscated, as the sequential nature of the data was preserved, more features could still be created from the time feature. 19 more features were created using the existing dataset. A summary of all the features (including the created features) and the target variable for each bidder id is shown in Table 4.

| Column Name | Description |
|---|---|
| **bidder_id** | Unique ID of bidder |

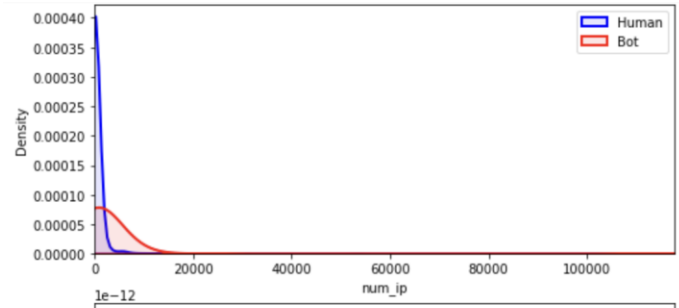
| | |
|---|---|
| **outcome** | Label of a bidder inidicating whether it is a robot. Value 1.0 indicated robot. Value 0.0 indicated human – Target variable |
| **num_bids** | Number of bids by bidder |
| **num_auct** | Number of auctions by bidder |
| **num_merch_type** | Number of merchandise categories bidded by bidder |
| **num_device_type** | Number of unique phone devices used by bidder |
| **num_time** | Num of different times bidded by bidder |
| **num_ctry** | Number of countries from which bidder's IP address belongs to |
| **num_id** | Number of IP addresses for bidder |
| **num_url** | Number of url for bidder |
| **mean_diff** | Mean time difference between bids by bidder |
| **std_diff** | Standard deviation of time difference between bids by bidder |
| **min_diff** | Minimum time difference between bids by bidder |
| **median_diff** | Median time difference between bids by bidder |
| **max_diff** | Max time difference between bids by bidder |
| **iqr_diff** | Interquartile range of time difference between bids by bidder |
| **num_fast_bids** | Number of bids by bidder with time difference of zero between bids |
| **num_first_bids** | Number of bids made first in an auction by bidder |
| **num_last_bid** | Number of bids made last in an auction by bidder |
| **max_bids_in_auct** | Max number of bids made in an auction by a bidder |
| **max_bids_per_device** | Max number of bids per device made in an auction by a bidder |
| **max_bids_per_device_per_auction** | Max number of bids per device per auction by a bidder |
| **percent_fast_bids** | Ratio of fast bids over total number of bids by bidder |
| **bids_per_auct** | Average bids made by bidder per auction |
| **bids_per_device** | Average bids made by bidder per device |
| **bids_per_url** | Average bids made by bidder per url |
| **devices_per_auction** | Average devices used by bidder per auction |
| **ips_per_country** | Average IP addresses used by bidder per country |
| **percent_max_bids** | Ratio of max bids over total number of bids by bidder |

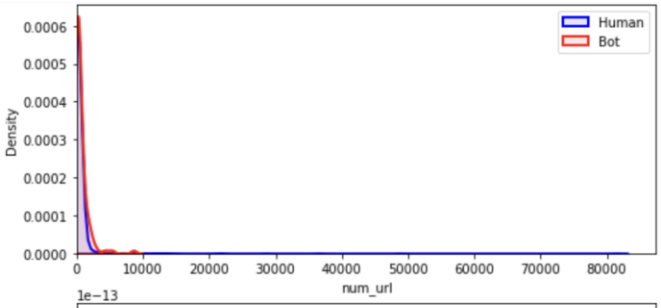
*Table 3: Summary table of total number of features*

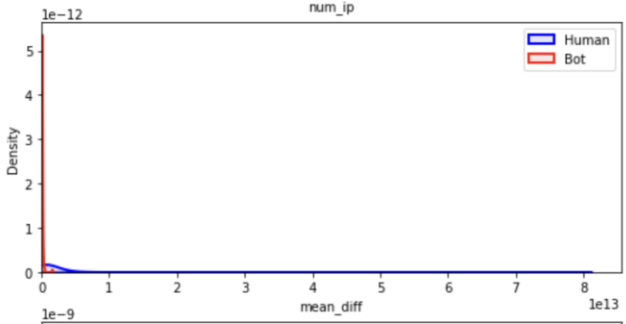
From Table 4, there are total of 27 features (with 1 target variable) for each bidder, on which exploratory data analysis will be conducted to only keep relevant features that will be able to differentiate a bot from a human.

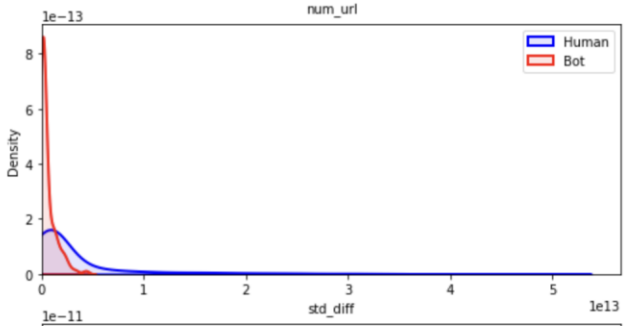
## Exploratory Data Analysis

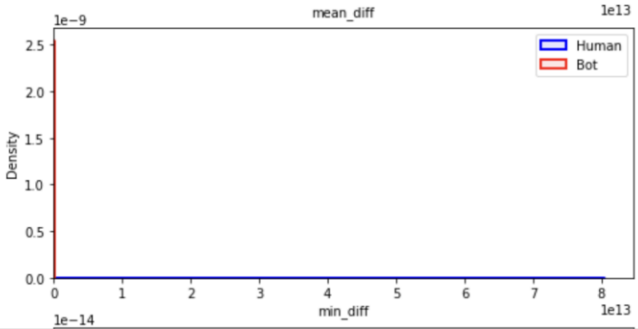
Kernel Density plots, which is a proxy for the probability distribution plots, for the 27 features differentiating between Human and bots from the feature engineered dataset is shown in Figure 2.

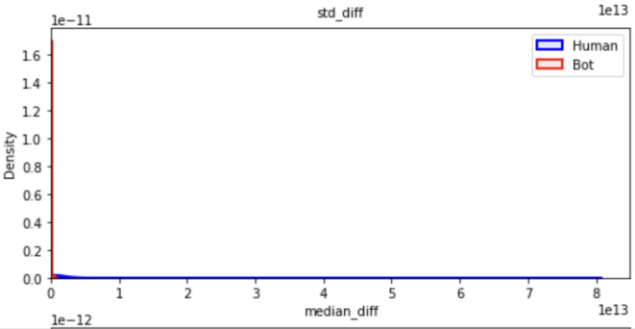

Human
Bot
Density
num_ip

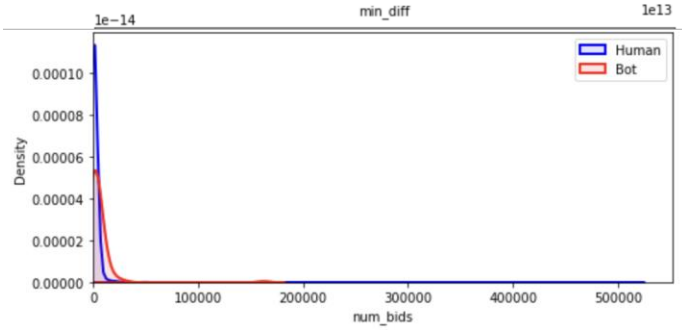

Human
Bot
Density
num_url

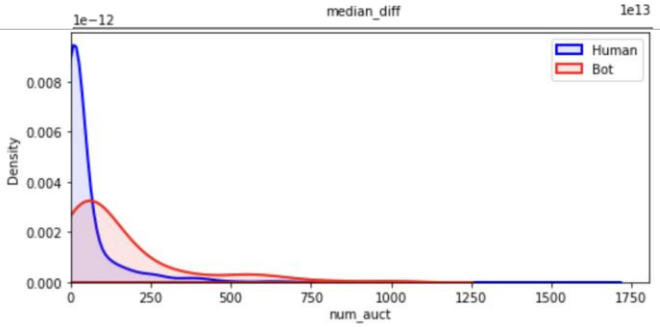

1e−12
Human
Bot
Density
mean_diff
1e13

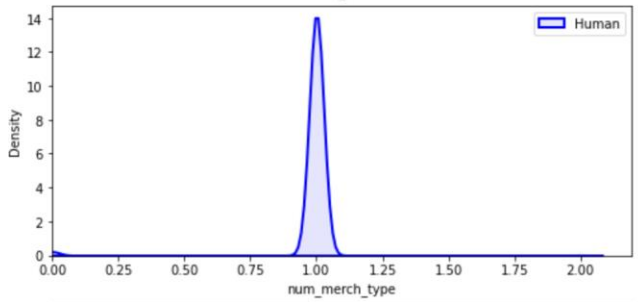

1e−13
Human
Bot
Density
std_diff
1e13

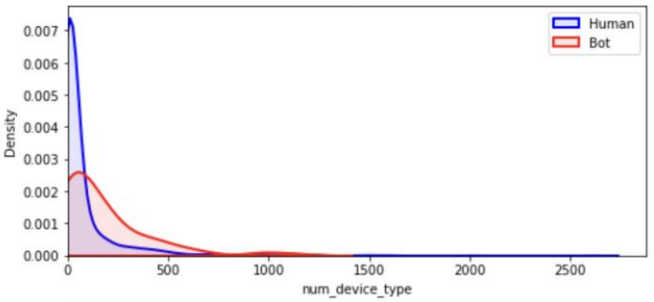

1e−9
Human
Bot
Density
min_diff
1e13

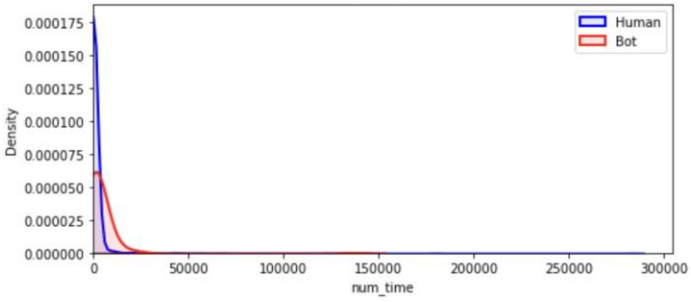

1e−11
Human
Bot
Density
median_diff
1e13

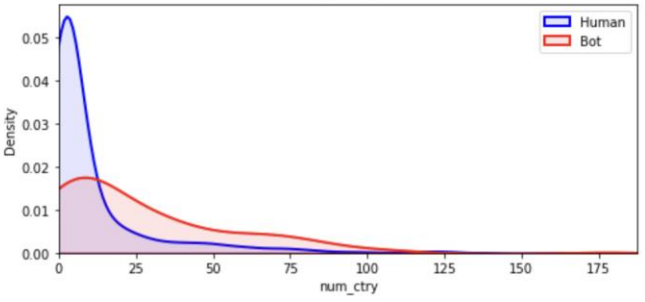

1e−14
Human
Bot
Density
num_bids

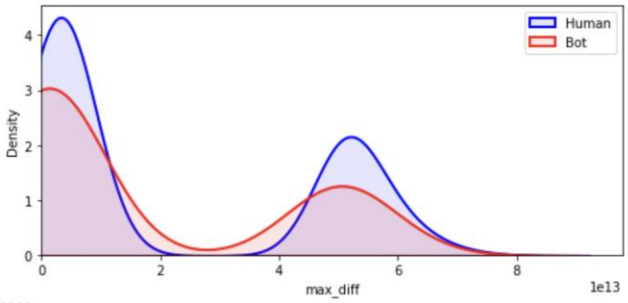

1e−12
Human
Bot
Density
num_auct

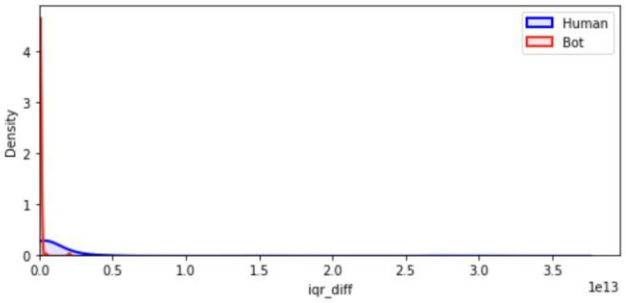

Human
Density
num_merch_type

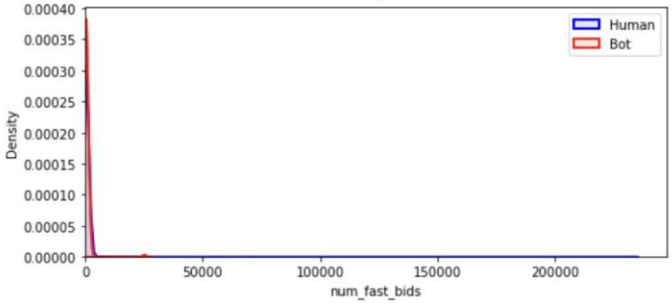

Human
Bot
Density
num_device_type

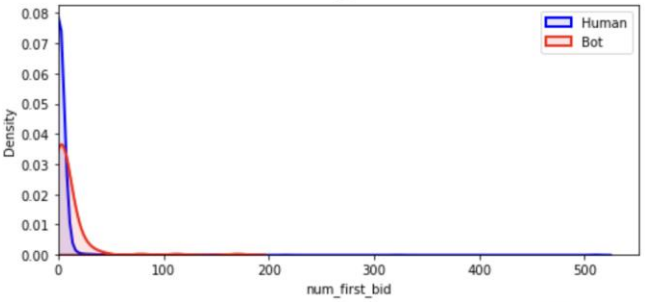

Human
Bot
Density
num_time

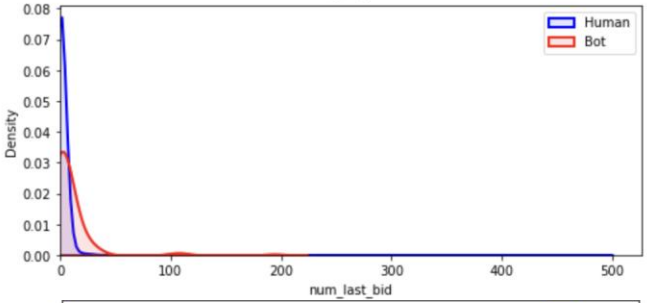

Human
Bot
Density
num_ctry

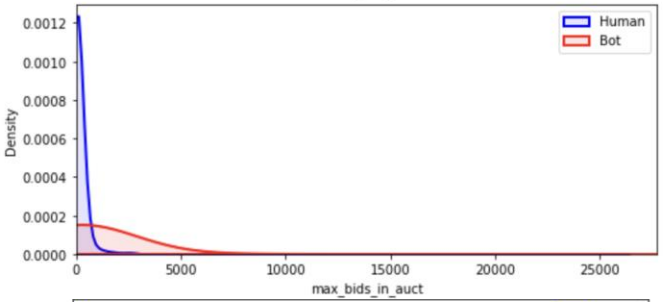

Human
Bot
Density
max_diff
1e13

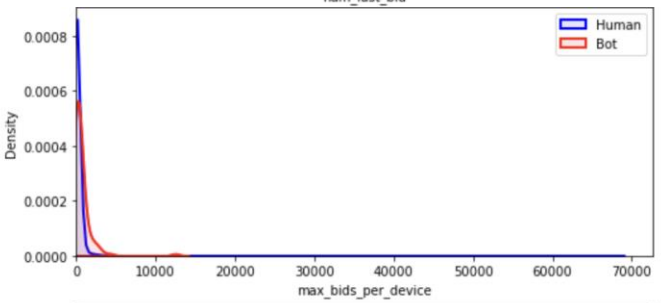

Human
Bot
Density
iqr_diff
1e13

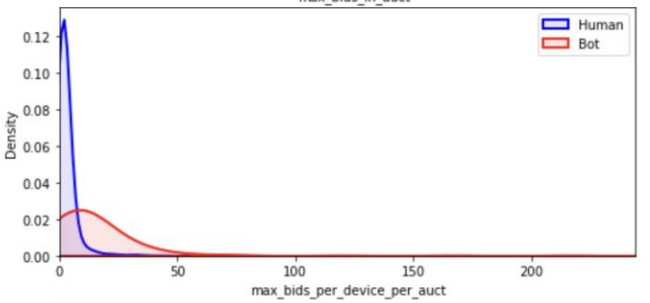

Human
Bot
Density
num_fast_bids

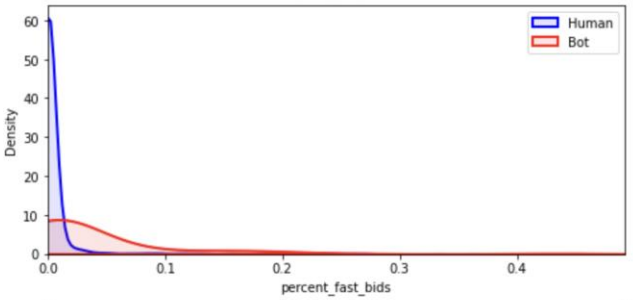

Human
Bot
Density
num_first_bid

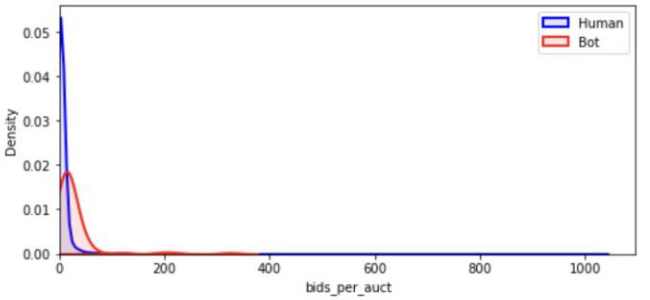

Human
Bot
Density
num_last_bid

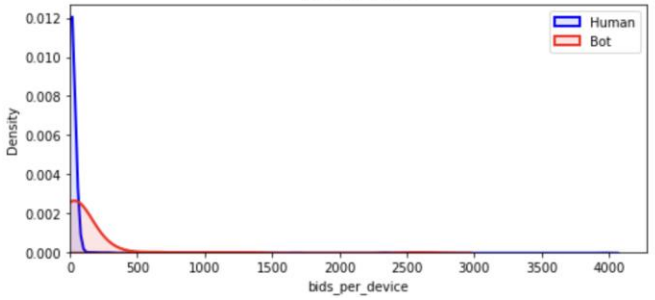

Human
Bot
Density
max_bids_in_auct

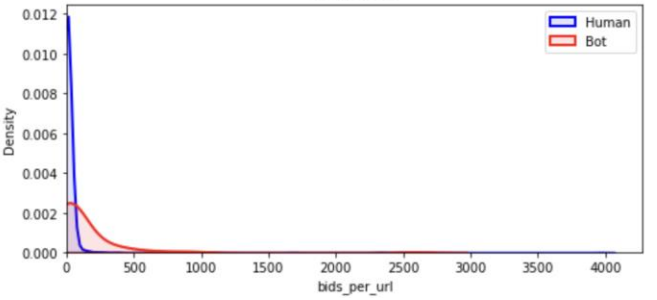

Human
Bot
Density
max_bids_per_device


Human
Bot
Density
max_bids_per_device_per_auct


Human
Bot
Density
percent_fast_bids


Human
Bot
Density
bids_per_auct


Human
Bot
Density
bids_per_device


Human
Bot
Density
bids_per_url

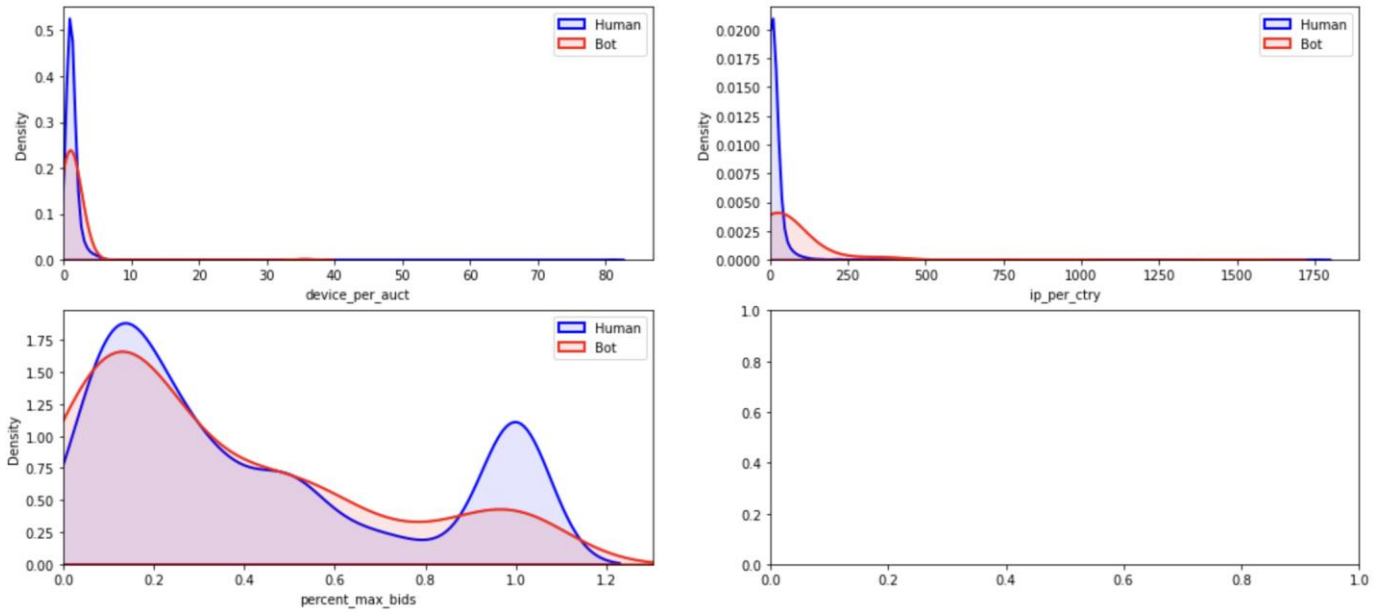



*Figure 2: Density plot of 27 features that differentiate between human and bots*

Based on the density plots shown above, it is observed that num_merch_type, num_fast_bids, max_bids_per_device and num_url is not able to differentiate between human and bots effectively. Therefore, these 4 features are dropped from the final dataset to be used for model training.

In addition, when the total number of bids by each bidder from the bot's data (when outcome variable equals to one) was analysed, it was deduced that there were 5 outliers in the data. Intuitively, a bot should have made a greater number of bids compared to a human. As shown in **Error! Reference source not found.** , these 5 outliers, which are bots, have made only 1 bid compared to other bots, where the next highest number of bids is 4 bids. Considering these 5 outlier bots will skew the results when training the dataset with models, these 5 outliers were removed from the final dataset.

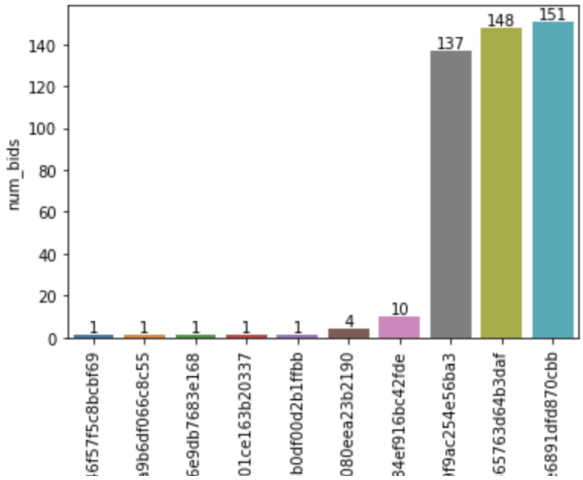



*Chart 3: 5 outlier bots that have only made 1 bid compared to the next highest bid of 4 bids a bot has made*

The final dataset to be used for predictive modelling of bots consists of 23 features and 1 target variable. The dataset contains 2008 unique bidders after removing the 5 outliers. This final unique bidders' dataset consists of 98 bots and 1910 humans.

## Cost-benefit matrix

| | | Predicted | |
|---|---|---|---|
| | | **P** | **N** |
| **Actual** | **P** | **0** | **-3** |
| | **N** | **-1** | **0** |

*Figure 3 4: Costs-Benefit Matrix*

Since businesses in the online auction platforms are losing 4% to 18% of their customer base due to presence of bots, an assumption is made that for every bot present in an auction that is not predicted as a bot by the model, 11% (average of 4% and 18%) of the bidders in the auction is lost from the customer base. Based on the current dataset given, there are ~25 bidders on average per auction. Therefore, for every bot that is falsely predicted to be a human, 3 human bidders are lost from the platform customer base. In addition, for every human bidder that is falsely identified as a bot, that bidder is banned from the platform. Therefore, we lose 1 customer from the customer base. Based on these assumptions, a cost-benefit matrix is shown above in fig.3. This cost-benefit matrix will be used as part of the expected value framework, which is the main performance metric used to identify the best model to predict bots from human bidders.

# Models & Results

Four Machine-Learning models (Logistic Regression, Support Vector Machines, Random Forest and XGBoost) were built to predict the bots from human bidders. The models were hyperparameter tuned using GridSearchCV library package with best expected value as the scoring method to identify the best parameters for each model that gives the highest expected value based on the cost-benefit matrix defined and the confusion matrix deduced from a threshold or decision function. The threshold is optimized for best EV value for each model as the dataset is highly imbalanced. Stratified cross-validation of 5-folds was used across all model during GridSearchCV. The threshold is optimized for best EV value for each model as the dataset is highly imbalanced and a default of 0.5 for threshold cannot be used. The models are then compared based on the best expected value to choose the best model to be used for business application.

## 1. Logistic Regression

### Introduction

Logistic regression, (or Logit model) is used for classification and predictive analytics. It estimates the probability of an event based on dataset of independent variables. The dependent variable which is a probability, ranges between 0 and 1. In logistic regression, a logit transformation is applied on the odds, i.e., probability of success divided by probability of failure. This is also known as log-odds.

The co-efficient in this model is normally determined via Maximum Likelihood Estimation (MLE). Once the optimal coefficient is found, the conditional probabilities for each observation can be calculated, logged and summed together to yield a predicted probability. For binary classification, a probability less than 0.5 will predict as 0 while probability above 0.5 is predicted as 1.

### Hyperparameter Tuning

GridSearchCV was used with “Penalty” and “C” parameters to find out the best model. The results are as below:

| Parameter | Description | Value |
|---|---|---|
| **‘l1’, ‘l2’** | Choice between Lasso and Ridge Regression | ‘l2’ |
| **‘C’** | Inverse of regularization strength | 1.0 |

*Table 4: Best parameter values for Logistic Regression*

Considering these, the best Expected Value score works to **-0.1190**.

### Model Results

With the GridSearchCV results, we ran the Logistic Regression model on our dataset with a pipeline initiated with Standard Scaler and LR Model. With a stratified train-test-split of 30% test size, the predictions by the model were used to assess precision, recall, f1 score and the expected value across different decision functions scores. The charts below indicate how the scores vary with the decision function score:

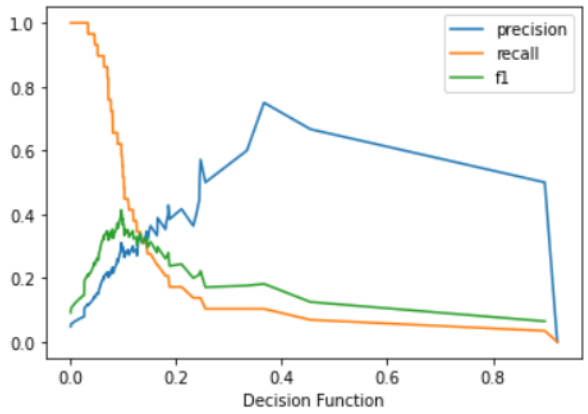



*Chart 4: Logistic Regression Model's Precision, Recall, F1 Curve*

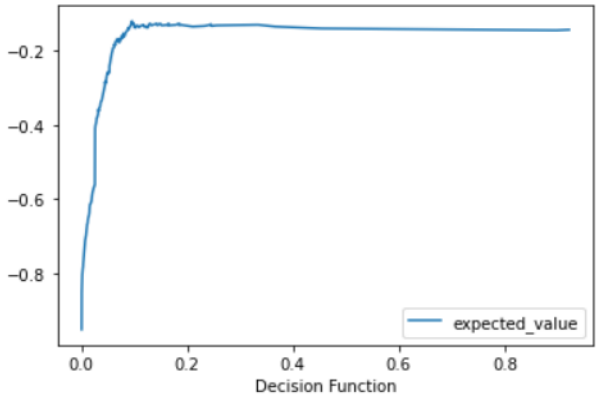



*Chart 5: Logistic Regression Model's Expected Value Curve*

The best threshold value is the threshold at which the expected value is the highest. The best score values are given below:

| | |
|---|---|
| Best Threshold Value | : 0.0954 |
| Best Expected Value | : -0.1211 |
| Best F1 Score | : 0.4045 |

## 2. Support Vector Machine

### Introduction

Support Vector Machine (SVM) is another effective model used for classification. SVM is preferred as it produces significant accuracy with less computational power. The objective of SVM is to find a hyperplane in an N-dimensional space (N: number of features) that classify the data. As many hyperplanes could exist, we choose the one maximum margin i.e., maximum distance between data points of both classes. The selected hyperplane would act as a decision boundary that classifies the data. Data points that lie on the margin or on the wrong side of the margin are called the **support vectors.** They "support" the decision boundary and the margin in the sense that moving these support vectors will change the decision boundary. These support vectors are restricted by a fixed "budget" or "tolerance of being on the wrong side. The larger the budget, larger is the tolerance and margin and vice-versa.

SVMs allow non-linear decision boundaries to classify non-linearly separated data. With (x1, x2, …, xp) features, nonlinear function *f(x1, x2, …, xp)* defines the decision function. The points where *f(x1, x2, …, xp)* = 0 defines the nonlinear function decision boundary in the (x1, x2, …, xp) space. We compute the decision function and compare against the threshold to predict the results.

### Hyperparameter tuning

GridSearchCV was used with "Linear" and "RBF" kernels to find out the best model. The results are below:

| Parameter | Description | Value |
|---|---|---|
| **SVM kernel** | Non-Linear function to define decision function | "RBF" (Radial Basis Function) |
| **SVM budget** | Maximum tolerance for support vector to be on the wrong side of the margin | 1.0 |

*Table 5: Best parameter values for Support Vector Machines*

Considering these, the best Expected Value score for the cross validation of the full dataset is **-0.0991**.

### Model Results

With the GridSearchCV results, we ran SVM on our dataset with a pipeline of two steps, Standard scalar and SVM model. With a stratified train-test-split of 30% test size, the predictions that the model gave on the test dataset were used to evaluate the precision, recall, f1 score and the expected value across varying decision functions scores. The two charts below show how these scores vary with the decision function score:

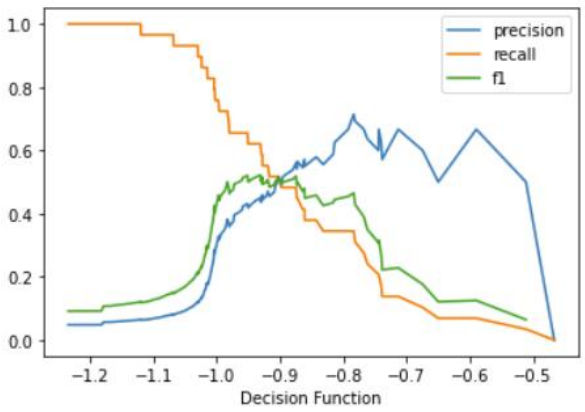



*Chart 6: SVM Model's Precision, Recall, F1 Curve*

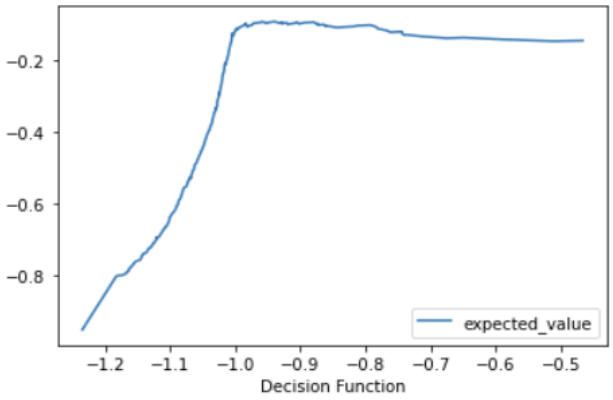



*Chart 7: SVM Model's Expected Value Curve*

The best decision function score is the one which results in maximum expected value score. The results are as below:

| | |
|---|---|
| Best decision function score | : -0.959 |
| Best expected value | : -0.0991 |
| Best f1 score | : 0.507 |

## 3. Random Forest

### Introduction

Random Forest model consists of many decision trees. Each decision tree selects 'm' features from the 23 features available to us. This reduces the variance that would usually come from a single decision tree. The Random Forest model is an effective classification model that outputs the majority value from all the decision trees. Our objective is to ensure that the Random Forest model predicts all the bots from our dataset.

### Hyperparameter tuning

To devise the best Random Forest model to operate on the full dataset, we made use of GridSearchCV to find the best combination of values for the following parameters:

| Parameters | Description | Value |
|---|---|---|
| **Min_samples_leaf** | Min number of datapoints allowed in the leaf node | **1** |
| **Max_depth** | Max number of levels in each decision tree | **8** |
| **Min_samples_split** | Min number of data points placed in a node before the node is split | **4** |

*Table 6: Best parameter values for Random Forest*

Considering these values, the best Expected Value score that the Model generates from the full dataset is **-0.0782**.

### Model Results

We used these parameters to create a Random Forest model that works on the training dataset, so as to compare across all models in this study. We then used this model to evaluate the precision, recall, f1 score and the expected value across varying thresholds that we get from the predictions made on the test dataset. The two charts below show how these scores vary with the threshold values:

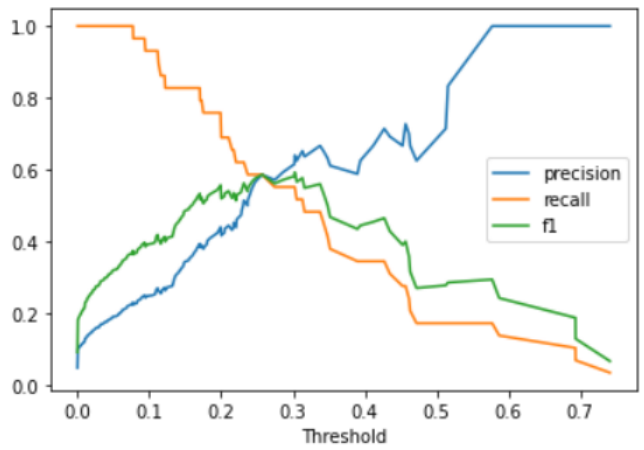



*Chart 8: Random Forest Model's Precision, Recall, F1 Curve*

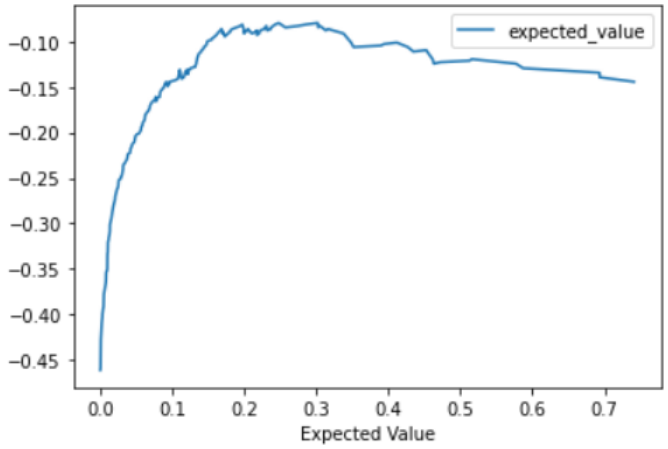



*Chart 9: Random Forest Model's Expected Value Curve*

We consider the best threshold value to be the threshold where the expected value to be the highest. Clearly, this also is the threshold where the f1 score of the model is the highest as well. The best score values are given below:

| | |
|---|---|
| Best threshold value | : 0.2477 |
| Best expected value | : -0.079 |
| Best f1 score | : 0.5762 |

### Interpretability

We made use of the Random Forest model to also understand how important each feature is to predicting if a customer is a bot or not. The below chart shows the feature importance for this study:

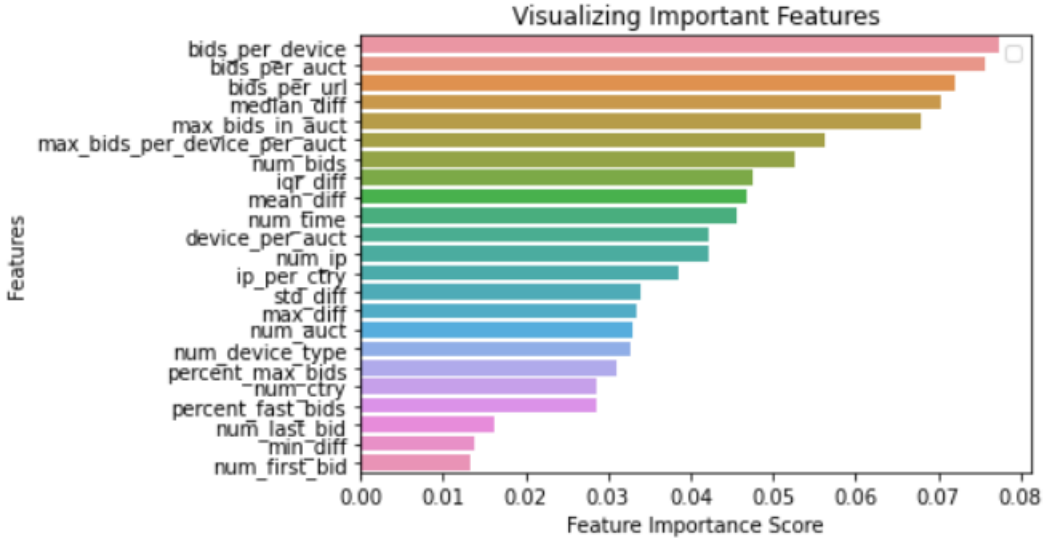



*Chart 10: Importance of each feature in the Random Forest Model*

From this chart, we see that 'bids_per_device' is the most important feature with a feature importance score of 7.7% and this contributes the most to predicting bots. This, intuitively, makes sense as we can consider a bot to gain more bids from a device as compared to a human. Out of the 23 features that we have considered in our study, seven features have a feature importance score greater than 5%.

## 4. XGBoost

### Introduction

XGBoost stands for e**X**treme **G**radient **B**oosting. It is a highly optimised (for running time) gradient-boosted decision tree-based classifier model. It provides parallel decision tree boosting using multiple decision tree models (similar to random forest model). The model trains by improving a single weak model by combining it with a number of other weak models in order to generate a collectively strong model. Gradient boosting sets targeted outcomes for the next

model in an effort to minimize errors. Targeted outcomes for each case are based on the gradient of the error with respect to the prediction. This model was trained using the dataset to predict bots from humans.

### Hyperparameter tuning

The best model is deduced by doing hyper parameter tuning using the GridSerachCV library package with learning rate, max depth, minimum child weight, subsample and no of estimators (number of trees) as the hyperparameters to be tuned:

| Parameters | Description | Value |
|---|---|---|
| **Learning rate** | Learning rate for the gradient boosting algorithm | **0.3** |
| **Max depth** | Maximum allowed depth of the trees | **6** |
| **Minimum child weight** | Minimum sum of weights needed in each child node for a split | **1** |
| **Subsample** | Subsample ratio from the training set. This means that for every tree a sub-selection of *samples* from the training set will be included into training by sampling without replacement | **1** |
| **No of estimators** | Number of trees | **150** |

*Table 7: Best parameter values for XGBoost*

Considering these values, the best Expected Value score that the Model generates from the K-fold validation of full dataset is **-0.0837**.

### Model Results

Using the optimized hyperparameters, an XGBoost model was built by training on a stratified training set of 70% of the data and the metrics deduced using the 30% of the test data to compare across all models in this study. The metrics deduced for this model are the precision, recall, f1 score and the expected value across varying thresholds that we obtain from the predictions made on the test dataset. The two charts below show how these scores vary with the threshold values:

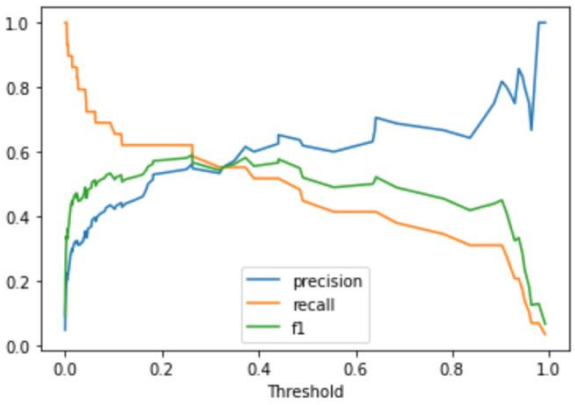



*Chart 11: XGBoost Model's Precision, Recall, F1 Curve*

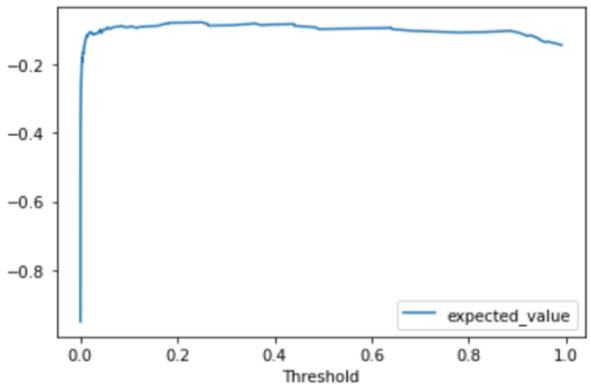



*Chart 12: XGBoost Model's Expected Value Curve*

We consider the best threshold value to be the threshold where the expected value to be the highest. Clearly, this also is the threshold where the f1 score of the model is the highest as well. The best score values are given below:

Best threshold : 0.2507

Best expected value : -0.0779

Best f1 score : 0.58

## Comparison of Model Results

| | Logistic Regression | Support Vector Machines | Random Forest | XGBoost |
|---|---|---|---|---|
| **Accuracy** | 0.912 | 0.942 | 0.959 | 0.957 |

| | | | | |
|---|---|---|---|---|
| **Precision** | 0.300 | 0.429 | 0.567 | 0.545 |
| **Recall** | 0.621 | 0.621 | 0.586 | 0.621 |
| **F1 Score** | 0.404 | 0.507 | 0.576 | 0.581 |
| ***Expected Value** | -0.1190 | -0.0991 | -0.0782 | -0.0837 |

*Table 8: Summary and comparison of results for each model*

The scores across Random Forest and XGBoost are relatively similar, but XGBoost leads with the scores. We can't categorically finalize on XGBoost given the similar scores. As we are focusing on the expected value, we can consider the Random Forest model for deployment.

*Expected value metric was deduced by training the model on the Stratified k-fold. For the rest of the metrics, stratified train-test-split was used.

# Business Application

For deployment as a business application, the optimized random forest model built (predict and treat model) must do better than a do-nothing model on the test set. This is to ensure that the deployed model creates more value for the online auction platform model by retaining its customer base to its best ability.

## Do nothing model

A do-nothing model is where we no prediction is done for identifying bots from human bidders on the online auction platform. This deduces that all the bots present are considered to be False negatives.

## Predict and Ban model

The predict and ban model utilizes the optimized random forest model that was trained using the dataset to predict which are the bidders are bots. Based on the model prediction, if a bidder is identified to be a bot, the bidder is banned from the online auction platform. However, the model may also ban human bidders predicted to be bots losing customers from customer base.

Based on the expected value framework, both models are compared (tested on the 30% test dataset) to deduce if the predict and treat model creates more value in retaining its customer base. The comparison table is shown below:

| Model | Expected Value |
|---|---|
| **Do-Nothing Model** | -0.0813 |
| **Predict and Ban Model** | -0.1443 |

Based on the results, the predict and ban model does give a better excepted value compared to the do-nothing model. Therefore, there is a good business case to deploy this model into production to ban identified bots present in the auction.

# Conclusion

The Bot Prediction Lifecyle will be able to help online auction platforms to identify bots effectively and to ban them from losing customer base and revenue. However, as depicted by the lifecycle, it is a continuous process to collect more data to train the different models to identify the best model to be deployed periodically. In addition, the business metrics (such as customer retention and identification of revenue loss due to bots) need to be also monitored and understood in a periodic manner if the model is effectively retaining customer base over time.
Lastly, care should also be taken to understand the maturity of a bidder's information before a bidder is included to be assessed using the model to be identified as a bot or not. This is to ensure that the model does not wrongly identify a human bidder as a bot due to lack of information.

Therefore, with these caveats in place, The Bot Prediction Lifecycle would largely assist online auction platform businesses to eradicate effectively the bots present in their auctions, retaining their customer base and increasing revenue consequently.

# References


- https://netacea.com/blog/what-are-bots-costing-businesses-in-2021/
- https://www.kasada.io/sniper-bots/
- https://www.nber.org/system/files/working_papers/w20942/w20942.pdf
- https://www.ibm.com/sg-en/topics/logistic-regression
- https://www.kaggle.com/c/facebook-recruiting-iv-human-or-bot

# Appendix

In [2]:
```
import pandas as pd

bids_df = pd.read_csv('data/bids.csv')
train_df = pd.read_csv('data/train.csv')
```

In [3]:
```
print(train_df.shape)
print(bids_df.shape)
```

```
(2013, 4)
(7656334, 9)
```

In [4]:
```
train_df.head()
```

Out[4]:

| | bidder_id | payment_account | address | outcome |
|---|---|---|---|---|
| **0** | 4a791121f1d9c9c192d13051301984388c491 | a3d2de7675556553a5f08e4c88d2c228754av | a3d2de7675556553a5f08e4c88d2c228vt0u4 | 0.0 |
| **1** | 6feeeab451fcc546e1c43867e04bd5d2294fc | a3d2de7675556553a5f08e4c88d2c228v1sga | ae87054e5a97a8f840a3991d12611fdcrfbq3 | 0.0 |
| **2** | 0bdfae6168380fa2e00853fd3d6199ba8ed7d | a3d2de7675556553a5f08e4c88d2c2280cybl | 92520288b50f03907041887884ba49c0cl0pd | 0.0 |
| **3** | 3b4586382b3ff164e8bad37b15a268a540996 | 51d80e233f7b6a7dfdee484a3c120f3b2ita8 | 4cb9717c8ad7e88a9a284989dd79b98dbevyi | 0.0 |
| **4** | 10c7f4bdc67a50bab856b9ca76a83ce980f96 | a3d2de7675556553a5f08e4c88d2c22857ddh | 2a96c3ce94b3be921e0296097b88b56a7x1ji | 0.0 |

In [5]:
```
bids_df.head()
```

Out[5]:

| | bid_id | bidder_id | auction | merchandise | device | time | country | ip | url |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | c6faf9e526c72bc8383f2bc4417a465d2fa27 | ewmzr | jewelry | phone0 | 9759243157894736 | us | 69.166.231.58 | vasstdc27m7nks3 |
| **1** | 1 | d96e93359ed70b7b4eebac163f448afeb50dc | aeqok | furniture | phone1 | 9759243157894736 | in | 50.201.125.84 | jmqlhflrzwuay9c |
| **2** | 2 | 065bdcc596308d1fdd0014261d6e198aa138e | wa00e | home goods | phone2 | 9759243157894736 | py | 112.54.208.157 | vasstdc27m7nks3 |
| **3** | 3 | e97aa310037dd210f52b5b164077a926bb5de | jefix | jewelry | phone4 | 9759243157894736 | in | 18.99.175.133 | vasstdc27m7nks3 |
| **4** | 4 | 41a7f56fcc5771d305baa6e989fb2e8cceb21 | jefix | jewelry | phone5 | 9759243157894736 | in | 145.138.5.37 | vasstdc27m7nks3 |

In [6]:
```
print(train_df.bidder_id.nunique())
print(bids_df.bidder_id.nunique())
```

```
2013
6614
```

In [7]:
```
train_bidder_ids = list(train_df.bidder_id)
bids_bidder_ids = list(bids_df.bidder_id.unique())

print(train_bidder_ids[:5])
```

```
['4a791121f1d9c9c192d13051301984388c491', '6feeeab451fcc546e1c43867e04bd5d2294fc', '0bdfae6168380fa2e00853fd3d6199ba8ed7d', '3b4
586382b3ff164e8bad37b15a268a540996', '10c7f4bdc67a50bab856b9ca76a83ce980f96']
```

In [8]:
```
not_found = []
for bidderid in train_bidder_ids:
    if bidderid not in bids_bidder_ids:
        not_found.append(bidderid)
print(len(not_found), 'bidder ids from train.csv are not found in bids.csv')
```

```
29 bidder ids from train.csv are not found in bids.csv
```

In [9]:
```
bidder_counts = bids_df.groupby("bidder_id")['time'].count().reset_index()
bidder_counts = bidder_counts.rename(columns={'time':'num_bids'})
bidder_counts
```

Out[9]:

| | bidder_id | num_bids |
|---|---|---|
| **0** | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 8 |
| **1** | 0014e3b911d1420e43ced1dc4fc18fde0fd0c | 7 |
| **2** | 002828f800c5132e297cfe3d44fbde9aeac51 | 2 |
| **3** | 00402f50c4086f320cb6bf94f04462ea32441 | 249 |
| **4** | 0040cea6b93afd86768c365d89513ffb7c0ba | 31 |
| **...** | ... | ... |
| **6609** | ffd49be672b1ba493b07dccf29311045c5392 | 168 |
| **6610** | ffdd8ed91a683b0f2a8237798ebe9214e3c43 | 5 |
| **6611** | ffde8717e9a547d536a4e9c3f44782829c021 | 24 |
| **6612** | ffe66dcb4b34bdbb5e17d7db7b1395e3fbbb7 | 3 |
| **6613** | ffef2a34068ab5328795bc9ed2da46aa29626 | 103 |

6614 rows × 2 columns

In [10]:
```
feature_set = train_df.merge(bidder_counts, on='bidder_id', how='left')  # Make sure NOT to use the default inner join (how='inne
print(feature_set.isnull().sum())
```

```
feature_set['num_bids'] = feature_set['num_bids'].fillna(0)  # fill nas with zeros
feature_set.isnull().sum()
```

```
bidder_id           0
payment_account     0
address             0
outcome             0
num_bids           29
dtype: int64
```

Out[10]:
```
bidder_id          0
payment_account    0
address            0
outcome            0
num_bids           0
dtype: int64
```

In [11]:
```
bidder_unique = bids_df.groupby("bidder_id").nunique()
bidder_unique
```

Out[11]:

| bidder_id | bid_id | auction | merchandise | device | time | country | ip | url |
|---|---|---|---|---|---|---|---|---|
| **0013f9b8d1f462df4462e1c1216e441ba6e18** | 8 | 6 | 1 | 7 | 8 | 2 | 8 | 4 |
| **0014e3b911d1420e43ced1dc4fc18fde0fd0c** | 7 | 7 | 1 | 7 | 7 | 4 | 7 | 4 |
| **002828f800c5132e297cfe3d44fbde9aeac51** | 2 | 2 | 1 | 1 | 2 | 1 | 2 | 1 |
| **00402f50c4086f320cb6bf94f04462ea32441** | 249 | 89 | 1 | 99 | 249 | 21 | 151 | 98 |
| **0040cea6b93afd86768c365d89513ffb7c0ba** | 31 | 23 | 1 | 21 | 31 | 6 | 28 | 4 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **ffd49be672b1ba493b07dccf29311045c5392** | 168 | 20 | 1 | 69 | 168 | 5 | 102 | 40 |
| **ffdd8ed91a683b0f2a8237798ebe9214e3c43** | 5 | 4 | 1 | 3 | 5 | 1 | 4 | 2 |
| **ffde8717e9a547d536a4e9c3f44782829c021** | 24 | 16 | 1 | 20 | 24 | 6 | 24 | 4 |
| **ffe66dcb4b34bdbb5e17d7db7b1395e3fbbb7** | 3 | 3 | 1 | 2 | 3 | 2 | 3 | 2 |
| **ffef2a34068ab5328795bc9ed2da46aa29626** | 103 | 70 | 1 | 50 | 103 | 13 | 73 | 33 |

6614 rows × 8 columns

In [17]:
```
#check average unique bidders per auction
bid_auction_unique = bids_df[['bidder_id','auction']].groupby("auction").nunique()
print(bid_auction_unique.mean())
```

```
bidder_id    25.40303
dtype: float64
```

In [18]:
```
bidder_unique = bidder_unique.reset_index()
bidder_unique
```

Out[18]:

| | bidder_id | bid_id | auction | merchandise | device | time | country | ip | url |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 8 | 6 | 1 | 7 | 8 | 2 | 8 | 4 |
| **1** | 0014e3b911d1420e43ced1dc4fc18fde0fd0c | 7 | 7 | 1 | 7 | 7 | 4 | 7 | 4 |
| **2** | 002828f800c5132e297cfe3d44fbde9aeac51 | 2 | 2 | 1 | 1 | 2 | 1 | 2 | 1 |
| **3** | 00402f50c4086f320cb6bf94f04462ea32441 | 249 | 89 | 1 | 99 | 249 | 21 | 151 | 98 |
| **4** | 0040cea6b93afd86768c365d89513ffb7c0ba | 31 | 23 | 1 | 21 | 31 | 6 | 28 | 4 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **6609** | ffd49be672b1ba493b07dccf29311045c5392 | 168 | 20 | 1 | 69 | 168 | 5 | 102 | 40 |
| **6610** | ffdd8ed91a683b0f2a8237798ebe9214e3c43 | 5 | 4 | 1 | 3 | 5 | 1 | 4 | 2 |
| **6611** | ffde8717e9a547d536a4e9c3f44782829c021 | 24 | 16 | 1 | 20 | 24 | 6 | 24 | 4 |
| **6612** | ffe66dcb4b34bdbb5e17d7db7b1395e3fbbb7 | 3 | 3 | 1 | 2 | 3 | 2 | 3 | 2 |
| **6613** | ffef2a34068ab5328795bc9ed2da46aa29626 | 103 | 70 | 1 | 50 | 103 | 13 | 73 | 33 |

6614 rows × 9 columns

In [19]:
```
train_set = train_df.merge(bidder_unique, on='bidder_id', how='left').drop(columns = ['payment_account', 'address'], errors = 'ig
train_set = train_set.rename(columns = {'bid_id': 'num_bids', 'auction': 'num_auct', 'merchandise': 'num_merch_type', 'device':
train_set = train_set.fillna(0)
train_set.isnull().sum()
```

Out[19]:
```
bidder_id          0
outcome            0
num_bids           0
num_auct           0
num_merch_type     0
num_device_type    0
num_time           0
num_ctry           0
num_ip             0
num_url            0
dtype: int64
```

In [20]:
```
train_set
```

Out[20]:

| | bidder_id | outcome | num_bids | num_auct | num_merch_type | num_device_type | num_time | num_ctry | num_ip | num_ |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 4a791121f1d9c9c192d13051301984388c491 | 0.0 | 24.0 | 18.0 | 1.0 | 14.0 | 24.0 | 6.0 | 20.0 | |
| **1** | 6feeeab451fcc546e1c43867e04bd5d2294fc | 0.0 | 3.0 | 1.0 | 1.0 | 2.0 | 3.0 | 1.0 | 3.0 | |
| **2** | 0bdfae6168380fa2e00853fd3d6199ba8ed7d | 0.0 | 4.0 | 4.0 | 1.0 | 2.0 | 4.0 | 1.0 | 4.0 | |
| **3** | 3b4586382b3ff164e8bad37b15a268a540996 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **4** | 10c7f4bdc67a50bab856b9ca76a83ce980f96 | 0.0 | 155.0 | 23.0 | 1.0 | 53.0 | 155.0 | 2.0 | 123.0 | 9 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2008** | c51d8ffc9d930dc9a5415dc46120f49c88267 | 0.0 | 36.0 | 25.0 | 1.0 | 4.0 | 33.0 | 4.0 | 5.0 | |
| **2009** | 6118a86e8b9eddcfb712642cc966f46de217a | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **2010** | 4944331d51a39cbe882cd9f5fc11f4fffa2c0 | 0.0 | 2.0 | 1.0 | 1.0 | 2.0 | 2.0 | 1.0 | 2.0 | |
| **2011** | 2ac59d862a484670bf4c4a8abd4161a955b38 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **2012** | 8e591eea755db2f07e1bb1f749eec5d191ab7 | 0.0 | 2.0 | 1.0 | 1.0 | 1.0 | 2.0 | 1.0 | 1.0 | |

2013 rows × 10 columns

In [101…
```
len(train_set[train_set['outcome'] == 1])
```

Out[101]:
```
98
```

In [21]:
```
#creating time features
time_df = bids_df.sort_values(['bidder_id', 'time'])
time_df.head(10)
```

Out[21]:

| | bid_id | bidder_id | auction | merchandise | device | time | country | ip | |
|---|---|---|---|---|---|---|---|---|---|
| **438339** | 438339 | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 2dfh7 | jewelry | phone252 | 9761753578947368 | in | 109.120.101.102 | j9r15qhlmx1( |
| **739754** | 739754 | 0013f9b8d1f462df4462e1c1216e441ba6e18 | befuo | jewelry | phone167 | 9763626789473684 | in | 249.82.135.248 | vasstdc27m7 |
| **784233** | 784233 | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 8hp9m | jewelry | phone1 | 9763837789473684 | in | 238.201.42.88 | j9r15qhlmx1( |
| **1079438** | 1079438 | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 2fdc6 | jewelry | phone20 | 9764985631578947 | us | 82.73.208.210 | eb7xzuinezym |
| **1098126** | 1098126 | 0013f9b8d1f462df4462e1c1216e441ba6e18 | toxfq | jewelry | phone35 | 9765134105263157 | in | 177.111.118.89 | y4n752u1rvı |
| **1430138** | 1430138 | 0013f9b8d1f462df4462e1c1216e441ba6e18 | lae7k | jewelry | phone71 | 9767696210526315 | in | 11.95.119.189 | j9r15qhlmx1( |
| **1588558** | 1588558 | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 8hp9m | jewelry | phone45 | 9768470947368421 | in | 201.170.80.59 | vasstdc27m7 |
| **1610781** | 1610781 | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 8hp9m | jewelry | phone71 | 9768530000000000 | in | 42.61.162.189 | j9r15qhlmx1( |
| **248666** | 248666 | 0014e3b911d1420e43ced1dc4fc18fde0fd0c | aeqok | home goods | phone1557 | 9760108210526315 | in | 177.158.106.58 | ylcz2qov0wu6 |
| **517210** | 517210 | 0014e3b911d1420e43ced1dc4fc18fde0fd0c | khgus | home goods | phone25 | 9762191894736842 | in | 53.230.178.66 | vasstdc27m7 |

In [22]:
```
#Creating time difference for each bidder and dropping nan values
firstdiff = time_df.groupby('bidder_id')[['time']].diff()
time_df['first_diff'] = firstdiff
firstdiff_feat = time_df[['bidder_id', 'first_diff']].dropna()
firstdiff_feat.head(10)
```

Out[22]:

| | bidder_id | first_diff |
|---|---|---|
| **739754** | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 1.873211e+12 |
| **784233** | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 2.110000e+11 |
| **1079438** | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 1.147842e+12 |
| **1098126** | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 1.484737e+11 |
| **1430138** | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 2.562105e+12 |
| **1588558** | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 7.747368e+11 |
| **1610781** | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 5.905263e+10 |
| **517210** | 0014e3b911d1420e43ced1dc4fc18fde0fd0c | 2.083684e+12 |
| **742654** | 0014e3b911d1420e43ced1dc4fc18fde0fd0c | 1.450737e+12 |
| **1361703** | 0014e3b911d1420e43ced1dc4fc18fde0fd0c | 3.622737e+12 |

In [23]:
```
#Create mean, median, std, min, max, interquartile range for the time difference for each bidder
bid_intervals = firstdiff_feat.groupby('bidder_id')[['first_diff']].describe().reset_index()
bid_intervals = bid_intervals.droplevel(axis=1, level=0)
bid_intervals = bid_intervals.rename(columns = {'': 'bidder_id', 'mean': 'mean_diff', 'std': 'std_diff', '50%': 'median_diff', 'r
bid_intervals['iqr_diff'] = bid_intervals['75%'] - bid_intervals['25%']
bid_intervals = bid_intervals.drop(['25%', '75%', 'count'], axis = 1)
bid_intervals
```

Out[23]:

| | bidder_id | mean_diff | std_diff | min_diff | median_diff | max_diff | iqr_diff |
|---|---|---|---|---|---|---|---|
| **0** | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 9.680602e+11 | 9.575254e+11 | 5.905263e+10 | 7.747368e+11 | 2.562105e+12 | 1.330789e+12 |
| **1** | 0014e3b911d1420e43ced1dc4fc18fde0fd0c | 2.120026e+12 | 1.044379e+12 | 7.083684e+11 | 1.989842e+12 | 3.622737e+12 | 1.177842e+12 |
| **2** | 002828f800c5132e297cfe3d44fbde9aeac51 | 6.236816e+13 | 0.000000e+00 | 6.236816e+13 | 6.236816e+13 | 6.236816e+13 | 0.000000e+00 |
| **3** | 00402f50c4086f320cb6bf94f04462ea32441 | 5.466914e+10 | 7.332996e+10 | 5.263158e+07 | 2.442105e+10 | 3.827895e+11 | 6.607895e+10 |
| **4** | 0040cea6b93afd86768c365d89513ffb7c0ba | 2.548244e+12 | 9.100694e+12 | 8.421053e+08 | 6.717368e+11 | 5.043837e+13 | 1.315197e+12 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **5552** | ffd49be672b1ba493b07dccf29311045c5392 | 8.130728e+10 | 1.077236e+11 | 1.052632e+08 | 4.368421e+10 | 7.146842e+11 | 8.165789e+10 |
| **5553** | ffdd8ed91a683b0f2a8237798ebe9214e3c43 | 1.467118e+13 | 2.706620e+13 | 7.000000e+09 | 1.738868e+12 | 5.520000e+13 | 1.631363e+13 |
| **5554** | ffde8717e9a547d536a4e9c3f44782829c021 | 3.301009e+12 | 1.063259e+13 | 5.000000e+09 | 6.934737e+11 | 5.184437e+13 | 1.436658e+12 |
| **5555** | ffe66dcb4b34bdbb5e17d7db7b1395e3fbbb7 | 4.502632e+10 | 4.700399e+10 | 1.178947e+10 | 4.502632e+10 | 7.826316e+10 | 3.323684e+10 |
| **5556** | ffef2a34068ab5328795bc9ed2da46aa29626 | 6.668834e+10 | 1.208364e+11 | 4.736842e+08 | 2.718421e+10 | 9.198947e+11 | 6.061842e+10 |

5557 rows × 7 columns

In [24]:
```
#merge first difference agg data with train data set and fill missing values with median values
train_set = train_set.merge(bid_intervals, on='bidder_id', how='left')
train_set = train_set.fillna(train_set.median())
train_set
```

```
/var/folders/cf/_53pz7w174sb1d98fsh_vdd40000gn/T/ipykernel_19335/743592726.py:3: FutureWarning: Dropping of nuisance columns in
DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid
columns before calling the reduction.
  train_set = train_set.fillna(train_set.median())
```

Out[24]:

| | bidder_id | outcome | num_bids | num_auct | num_merch_type | num_device_type | num_time | num_ctry | num_ip | num_ |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 4a791121f1d9c9c192d13051301984388c491 | 0.0 | 24.0 | 18.0 | 1.0 | 14.0 | 24.0 | 6.0 | 20.0 | |
| **1** | 6feeeab451fcc546e1c43867e04bd5d2294fc | 0.0 | 3.0 | 1.0 | 1.0 | 2.0 | 3.0 | 1.0 | 3.0 | |
| **2** | 0bdfae6168380fa2e00853fd3d6199ba8ed7d | 0.0 | 4.0 | 4.0 | 1.0 | 2.0 | 4.0 | 1.0 | 4.0 | |
| **3** | 3b4586382b3ff164e8bad37b15a268a540996 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **4** | 10c7f4bdc67a50bab856b9ca76a83ce980f96 | 0.0 | 155.0 | 23.0 | 1.0 | 53.0 | 155.0 | 2.0 | 123.0 | 9 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2008** | c51d8ffc9d930dc9a5415dc46120f49c88267 | 0.0 | 36.0 | 25.0 | 1.0 | 4.0 | 33.0 | 4.0 | 5.0 | |
| **2009** | 6118a86e8b9eddcfb712642cc966f46de217a | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **2010** | 4944331d51a39cbe882cd9f5fc11f4fffa2c0 | 0.0 | 2.0 | 1.0 | 1.0 | 2.0 | 2.0 | 1.0 | 2.0 | |
| **2011** | 2ac59d862a484670bf4c4a8abd4161a955b38 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **2012** | 8e591eea755db2f07e1bb1f749eec5d191ab7 | 0.0 | 2.0 | 1.0 | 1.0 | 1.0 | 2.0 | 1.0 | 1.0 | |

2013 rows × 16 columns

In [25]:
```
# count the number of bids for bidder ids with time difference of 0 - maybe a feture to differentiate
# bots as they may react faster than humans
#merge this feature with train data set and fill the missing values with 0
fast_bids = firstdiff_feat[firstdiff_feat['first_diff'] == 0].groupby('bidder_id').count().reset_index()
```

```
fast_bids = fast_bids.rename(columns = {'first_diff': 'num_fast_bids'})
print(fast_bids.head())
train_set = train_set.merge(fast_bids, on='bidder_id', how='left').fillna(0)
train_set
```

```
                              bidder_id  num_fast_bids
0  00512db4ae953baed983a4bcfa335e7412013              2
1  00e35b781106a924cdd8c9bd5ba5d4c57f200              8
2  00eb32718a2af3bd08e664baf136dc29d3d7a              1
3  00f0194f4991a2b80c0f810bc8a2f6935dc58              2
4  00f6d97bb02cbc8ccc813f08ccc35d27f84f7           1761
```

Out[25]:

| | bidder_id | outcome | num_bids | num_auct | num_merch_type | num_device_type | num_time | num_ctry | num_ip | num_ |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 4a791121f1d9c9c192d13051301984388c491 | 0.0 | 24.0 | 18.0 | 1.0 | 14.0 | 24.0 | 6.0 | 20.0 | |
| **1** | 6feeeab451fcc546e1c43867e04bd5d2294fc | 0.0 | 3.0 | 1.0 | 1.0 | 2.0 | 3.0 | 1.0 | 3.0 | |
| **2** | 0bdfae6168380fa2e00853fd3d6199ba8ed7d | 0.0 | 4.0 | 4.0 | 1.0 | 2.0 | 4.0 | 1.0 | 4.0 | |
| **3** | 3b4586382b3ff164e8bad37b15a268a540996 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **4** | 10c7f4bdc67a50bab856b9ca76a83ce980f96 | 0.0 | 155.0 | 23.0 | 1.0 | 53.0 | 155.0 | 2.0 | 123.0 | 9 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2008** | c51d8ffc9d930dc9a5415dc46120f49c88267 | 0.0 | 36.0 | 25.0 | 1.0 | 4.0 | 33.0 | 4.0 | 5.0 | |
| **2009** | 6118a86e8b9eddcfb712642cc966f46de217a | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **2010** | 4944331d51a39cbe882cd9f5fc11f4fffa2c0 | 0.0 | 2.0 | 1.0 | 1.0 | 2.0 | 2.0 | 1.0 | 2.0 | |
| **2011** | 2ac59d862a484670bf4c4a8abd4161a955b38 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **2012** | 8e591eea755db2f07e1bb1f749eec5d191ab7 | 0.0 | 2.0 | 1.0 | 1.0 | 1.0 | 2.0 | 1.0 | 1.0 | |

2013 rows × 17 columns

In [26]:
```
#No of times the bidder is first in the bid
first_bid = bids_df.sort_values(['auction', 'time'])
first_bid = first_bid.groupby('auction').first().reset_index()
first_bid = first_bid.groupby('bidder_id').count()['bid_id'].reset_index()
first_bid = first_bid.rename(columns = {'bid_id': 'num_first_bid'})
first_bid
```

Out[26]:

| | bidder_id | num_first_bid |
|---|---|---|
| **0** | 00512db4ae953baed983a4bcfa335e7412013 | 6 |
| **1** | 00f6d97bb02cbc8ccc813f08ccc35d27f84f7 | 4 |
| **2** | 0102ba941b052d9caaedda7e296c7d91783b1 | 1 |
| **3** | 014b86ec9b2da9a13b73659349fbee1680bc7 | 15 |
| **4** | 015344a674985d95ff96689843a4ba27138ad | 1 |
| **...** | ... | ... |
| **1465** | ff61cd9eb23ffbbb4b08eb8c3328ad76331da | 1 |
| **1466** | ff67dc2cb176581eb93768f3d14943dc9da5a | 1 |
| **1467** | ff8631028eadc04ffcc22b09d94d69d6bc3cf | 2 |
| **1468** | ffcf256495c1334dda186fe094fdc4829ddde | 14 |
| **1469** | ffd49be672b1ba493b07dccf29311045c5392 | 1 |

1470 rows × 2 columns

In [27]:
```
#No of times the bidder is last in the bid
last_bid = bids_df.sort_values(['auction', 'time'], ascending = [True, False])
last_bid = last_bid.groupby('auction').first().reset_index()
last_bid = last_bid.groupby('bidder_id').count()['bid_id'].reset_index()
last_bid = last_bid.rename(columns = {'bid_id': 'num_last_bid'})
last_bid
```

Out[27]:

| | bidder_id | num_last_bid |
|---|---|---|
| **0** | 0014e3b911d1420e43ced1dc4fc18fde0fd0c | 1 |
| **1** | 00512db4ae953baed983a4bcfa335e7412013 | 4 |
| **2** | 00afa68b3c556fc61b45e46c38c4aae3fbc2e | 1 |
| **3** | 00eb32718a2af3bd08e664baf136dc29d3d7a | 1 |
| **4** | 00f6d97bb02cbc8ccc813f08ccc35d27f84f7 | 2 |
| **...** | ... | ... |
| **1422** | ff1fd4e9802c1b6e656136b4ea51ad6046fe8 | 21 |
| **1423** | ff61cd9eb23ffbbb4b08eb8c3328ad76331da | 2 |
| **1424** | ff8631028eadc04ffcc22b09d94d69d6bc3cf | 2 |
| **1425** | ffcf256495c1334dda186fe094fdc4829ddde | 24 |
| **1426** | ffd49be672b1ba493b07dccf29311045c5392 | 7 |

1427 rows × 2 columns

In [28]:
```
#merge first and last bid features
train_set = train_set.merge(first_bid, on='bidder_id', how='left').fillna(0)
train_set = train_set.merge(last_bid, on='bidder_id', how='left').fillna(0)
train_set
```

Out[28]:

| | bidder_id | outcome | num_bids | num_auct | num_merch_type | num_device_type | num_time | num_ctry | num_ip | num_ |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 4a791121f1d9c9c192d13051301984388c491 | 0.0 | 24.0 | 18.0 | 1.0 | 14.0 | 24.0 | 6.0 | 20.0 | |
| **1** | 6feeeab451fcc546e1c43867e04bd5d2294fc | 0.0 | 3.0 | 1.0 | 1.0 | 2.0 | 3.0 | 1.0 | 3.0 | |
| **2** | 0bdfae6168380fa2e00853fd3d6199ba8ed7d | 0.0 | 4.0 | 4.0 | 1.0 | 2.0 | 4.0 | 1.0 | 4.0 | |
| **3** | 3b4586382b3ff164e8bad37b15a268a540996 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **4** | 10c7f4bdc67a50bab856b9ca76a83ce980f96 | 0.0 | 155.0 | 23.0 | 1.0 | 53.0 | 155.0 | 2.0 | 123.0 | 9 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2008** | c51d8ffc9d930dc9a5415dc46120f49c88267 | 0.0 | 36.0 | 25.0 | 1.0 | 4.0 | 33.0 | 4.0 | 5.0 | |
| **2009** | 6118a86e8b9eddcfb712642cc966f46de217a | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **2010** | 4944331d51a39cbe882cd9f5fc11f4fffa2c0 | 0.0 | 2.0 | 1.0 | 1.0 | 2.0 | 2.0 | 1.0 | 2.0 | |
| **2011** | 2ac59d862a484670bf4c4a8abd4161a955b38 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **2012** | 8e591eea755db2f07e1bb1f749eec5d191ab7 | 0.0 | 2.0 | 1.0 | 1.0 | 1.0 | 2.0 | 1.0 | 1.0 | |

2013 rows × 19 columns

In [29]:
```
# max no of bids per auction - bots should make more bids
max_bids_in_auct = bids_df.groupby(['bidder_id', 'auction']).count()
max_bids_in_auct = max_bids_in_auct.reset_index()[['bidder_id', 'auction', 'bid_id']].rename(columns = {'bid_id': 'max_bids_in_au
max_bids_in_auct = max_bids_in_auct[['bidder_id', 'max_bids_in_auct']].groupby('bidder_id').max().reset_index()
max_bids_in_auct
```

Out[29]:

| | bidder_id | max_bids_in_auct |
|---|---|---|
| **0** | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 3 |
| **1** | 0014e3b911d1420e43ced1dc4fc18fde0fd0c | 1 |
| **2** | 002828f800c5132e297cfe3d44fbde9aeac51 | 1 |
| **3** | 00402f50c4086f320cb6bf94f04462ea32441 | 80 |
| **4** | 0040cea6b93afd86768c365d89513ffb7c0ba | 3 |
| **...** | ... | ... |
| **6609** | ffd49be672b1ba493b07dccf29311045c5392 | 49 |
| **6610** | ffdd8ed91a683b0f2a8237798ebe9214e3c43 | 2 |
| **6611** | ffde8717e9a547d536a4e9c3f44782829c021 | 4 |
| **6612** | ffe66dcb4b34bdbb5e17d7db7b1395e3fbbb7 | 1 |
| **6613** | ffef2a34068ab5328795bc9ed2da46aa29626 | 6 |

6614 rows × 2 columns

In [30]:
```
#merge max bids per auction with train data set
train_set = train_set.merge(max_bids_in_auct, on='bidder_id', how='left').fillna(0)
train_set
```

Out[30]:

| | bidder_id | outcome | num_bids | num_auct | num_merch_type | num_device_type | num_time | num_ctry | num_ip | num_ |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 4a791121f1d9c9c192d13051301984388c491 | 0.0 | 24.0 | 18.0 | 1.0 | 14.0 | 24.0 | 6.0 | 20.0 | |
| **1** | 6feeeab451fcc546e1c43867e04bd5d2294fc | 0.0 | 3.0 | 1.0 | 1.0 | 2.0 | 3.0 | 1.0 | 3.0 | |
| **2** | 0bdfae6168380fa2e00853fd3d6199ba8ed7d | 0.0 | 4.0 | 4.0 | 1.0 | 2.0 | 4.0 | 1.0 | 4.0 | |
| **3** | 3b4586382b3ff164e8bad37b15a268a540996 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **4** | 10c7f4bdc67a50bab856b9ca76a83ce980f96 | 0.0 | 155.0 | 23.0 | 1.0 | 53.0 | 155.0 | 2.0 | 123.0 | 9 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2008** | c51d8ffc9d930dc9a5415dc46120f49c88267 | 0.0 | 36.0 | 25.0 | 1.0 | 4.0 | 33.0 | 4.0 | 5.0 | |
| **2009** | 6118a86e8b9eddcfb712642cc966f46de217a | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **2010** | 4944331d51a39cbe882cd9f5fc11f4fffa2c0 | 0.0 | 2.0 | 1.0 | 1.0 | 2.0 | 2.0 | 1.0 | 2.0 | |
| **2011** | 2ac59d862a484670bf4c4a8abd4161a955b38 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **2012** | 8e591eea755db2f07e1bb1f749eec5d191ab7 | 0.0 | 2.0 | 1.0 | 1.0 | 1.0 | 2.0 | 1.0 | 1.0 | |

2013 rows × 20 columns

In [31]:
```
#Max no of bids using the same device by a bidder
device = bids_df.groupby(['bidder_id', 'device']).nunique()[['bid_id', 'auction']]
device = device.reset_index().rename(columns = {'bid_id': 'max_bids_per_device', 'auction': 'num_auct_per_device'})
device = device.groupby('bidder_id').max().reset_index()
device['max_bids_per_device_per_auct'] = device['max_bids_per_device'] / device['num_auct_per_device']
device = device[['bidder_id', 'max_bids_per_device', 'max_bids_per_device_per_auct']]
device.head()
```

Out[31]:

| | bidder_id | max_bids_per_device | max_bids_per_device_per_auct |
|---|---|---|---|
| **0** | 0013f9b8d1f462df4462e1c1216e441ba6e18 | 2 | 1.000000 |
| **1** | 0014e3b911d1420e43ced1dc4fc18fde0fd0c | 1 | 1.000000 |
| **2** | 002828f800c5132e297cfe3d44fbde9aeac51 | 2 | 1.000000 |
| **3** | 00402f50c4086f320cb6bf94f04462ea32441 | 33 | 2.357143 |
| **4** | 0040cea6b93afd86768c365d89513ffb7c0ba | 4 | 1.333333 |

In [32]:
```
#merge with train data set
train_set = train_set.merge(device, on='bidder_id', how='left').fillna(0)
train_set
```

Out[32]:

| | bidder_id | outcome | num_bids | num_auct | num_merch_type | num_device_type | num_time | num_ctry | num_ip | num_ |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 4a791121f1d9c9c192d13051301984388c491 | 0.0 | 24.0 | 18.0 | 1.0 | 14.0 | 24.0 | 6.0 | 20.0 | |
| **1** | 6feeeab451fcc546e1c43867e04bd5d2294fc | 0.0 | 3.0 | 1.0 | 1.0 | 2.0 | 3.0 | 1.0 | 3.0 | |
| **2** | 0bdfae6168380fa2e00853fd3d6199ba8ed7d | 0.0 | 4.0 | 4.0 | 1.0 | 2.0 | 4.0 | 1.0 | 4.0 | |
| **3** | 3b4586382b3ff164e8bad37b15a268a540996 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **4** | 10c7f4bdc67a50bab856b9ca76a83ce980f96 | 0.0 | 155.0 | 23.0 | 1.0 | 53.0 | 155.0 | 2.0 | 123.0 | 9 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2008** | c51d8ffc9d930dc9a5415dc46120f49c88267 | 0.0 | 36.0 | 25.0 | 1.0 | 4.0 | 33.0 | 4.0 | 5.0 | |
| **2009** | 6118a86e8b9eddcfb712642cc966f46de217a | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **2010** | 4944331d51a39cbe882cd9f5fc11f4fffa2c0 | 0.0 | 2.0 | 1.0 | 1.0 | 2.0 | 2.0 | 1.0 | 2.0 | |
| **2011** | 2ac59d862a484670bf4c4a8abd4161a955b38 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **2012** | 8e591eea755db2f07e1bb1f749eec5d191ab7 | 0.0 | 2.0 | 1.0 | 1.0 | 1.0 | 2.0 | 1.0 | 1.0 | |

2013 rows × 22 columns

In [33]:
```
#percent of fast bids from total no of bids
import numpy as np
train_set['percent_fast_bids'] = train_set['num_fast_bids'] / train_set['num_bids']
# bids per auction
train_set['bids_per_auct'] = train_set['num_bids'] / train_set['num_auct']
#buds per device
train_set['bids_per_device'] = train_set['num_bids'] / train_set['num_device_type']
#bids per url
train_set['bids_per_url'] = train_set['num_bids'] / train_set['num_url']
#devices per auction
train_set['device_per_auct'] = train_set['num_device_type'] / train_set['num_auct']
#ips per country
train_set['ip_per_ctry'] = train_set['num_ip'] / train_set['num_ctry']
#percentage of max bids from no of bids
train_set['percent_max_bids'] = train_set['max_bids_per_device'] / train_set['num_bids']

train_set = train_set.fillna(0)
```

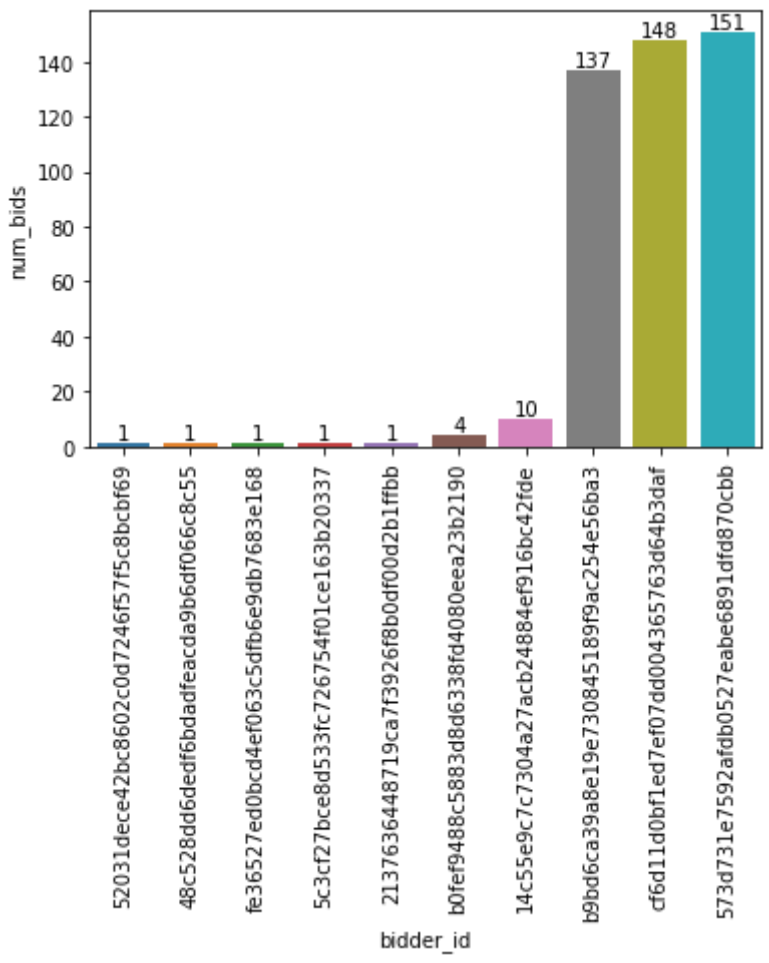
```
train_set.replace([np.inf, -np.inf], 0, inplace=True)
train_set
```

Out[33]:

| | bidder_id | outcome | num_bids | num_auct | num_merch_type | num_device_type | num_time | num_ctry | num_ip | num_ |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 4a791121f1d9c9c192d13051301984388c491 | 0.0 | 24.0 | 18.0 | 1.0 | 14.0 | 24.0 | 6.0 | 20.0 | |
| **1** | 6feeeab451fcc546e1c43867e04bd5d2294fc | 0.0 | 3.0 | 1.0 | 1.0 | 2.0 | 3.0 | 1.0 | 3.0 | |
| **2** | 0bdfae6168380fa2e00853fd3d6199ba8ed7d | 0.0 | 4.0 | 4.0 | 1.0 | 2.0 | 4.0 | 1.0 | 4.0 | |
| **3** | 3b4586382b3ff164e8bad37b15a268a540996 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **4** | 10c7f4bdc67a50bab856b9ca76a83ce980f96 | 0.0 | 155.0 | 23.0 | 1.0 | 53.0 | 155.0 | 2.0 | 123.0 | 9 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2008** | c51d8ffc9d930dc9a5415dc46120f49c88267 | 0.0 | 36.0 | 25.0 | 1.0 | 4.0 | 33.0 | 4.0 | 5.0 | |
| **2009** | 6118a86e8b9eddcfb712642cc966f46de217a | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **2010** | 4944331d51a39cbe882cd9f5fc11f4fffa2c0 | 0.0 | 2.0 | 1.0 | 1.0 | 2.0 | 2.0 | 1.0 | 2.0 | |
| **2011** | 2ac59d862a484670bf4c4a8abd4161a955b38 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **2012** | 8e591eea755db2f07e1bb1f749eec5d191ab7 | 0.0 | 2.0 | 1.0 | 1.0 | 1.0 | 2.0 | 1.0 | 1.0 | |

2013 rows × 29 columns

In [36]:
```
#removing outliers
import matplotlib.pyplot as plt
import seaborn as sns
fig, ax = plt.subplots()
num_bids_per_bot = sns.barplot(data = train_set[train_set['outcome'] == 1].sort_values('num_bids').head(10),

                               x = 'bidder_id',
                               y = 'num_bids',
                               ax = ax
                              )
ax.bar_label(ax.containers[0])
plt.xticks(rotation = 90)

plt.show()
```




In [37]:
```
train_set[train_set['outcome'] == 1].sort_values('num_bids').head(6)
```

Out[37]:

| | bidder_id | outcome | num_bids | num_auct | num_merch_type | num_device_type | num_time | num_ctry | num_ip | num_ |
|---|---|---|---|---|---|---|---|---|---|---|
| **615** | 52031dece42bc8602c0d7246f57f5c8bcbf69 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **775** | 48c528dd6dedf6bdadfeacda9b6df066c8c55 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **392** | fe36527ed0bcd4ef063c5dfb6e9db7683e168 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **1669** | 5c3cf27bce8d533fc726754f01ce163b20337 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **1102** | 2137636448719ca7f3926f8b0df00d2b1ffbb | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| **148** | b0fef9488c5883d8d6338fd4080eea23b2190 | 1.0 | 4.0 | 2.0 | 1.0 | 3.0 | 4.0 | 2.0 | 3.0 | |

6 rows × 29 columns

In [38]:
```
train_set = train_set.drop([615, 775, 392, 1669, 1102], axis = 0)
```

In [39]:
```
train_set.groupby('outcome').mean().T
```

Out[39]:

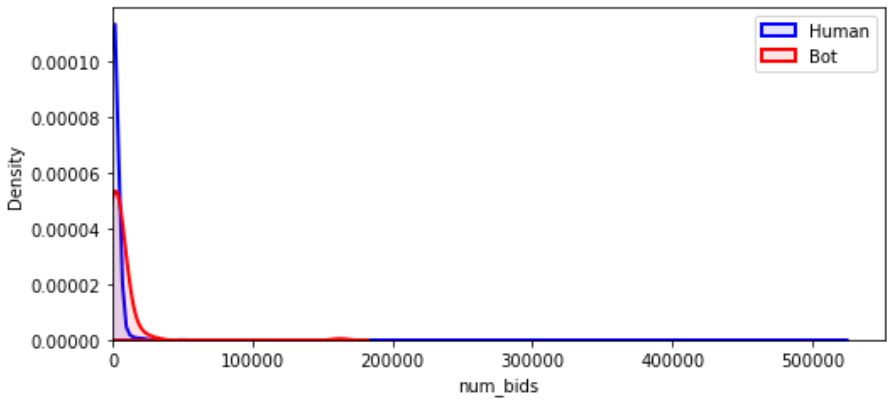
| outcome | 0.0 | 1.0 |
|---|---|---|
| **num_bids** | 1.392046e+03 | 4.208276e+03 |
| **num_auct** | 5.718901e+01 | 1.523878e+02 |
| **num_merch_type** | 9.853403e-01 | 1.000000e+00 |
| **num_device_type** | 7.282461e+01 | 1.719082e+02 |
| **num_time** | 1.157949e+03 | 3.857806e+03 |
| **num_ctry** | 1.239634e+01 | 2.761224e+01 |
| **num_ip** | 5.724309e+02 | 2.509571e+03 |
| **num_url** | 3.300979e+02 | 5.723163e+02 |
| **mean_diff** | 2.915060e+12 | 5.332332e+10 |
| **std_diff** | 4.195499e+12 | 5.086255e+11 |
| **min_diff** | 9.789134e+11 | 8.002148e+07 |
| **median_diff** | 1.662292e+12 | 1.228330e+10 |
| **max_diff** | 2.158563e+13 | 1.602854e+13 |
| **iqr_diff** | 1.515340e+12 | 3.669294e+10 |
| **num_fast_bids** | 2.340974e+02 | 3.504694e+02 |
| **num_first_bid** | 2.603141e+00 | 8.295918e+00 |
| **num_last_bid** | 2.627225e+00 | 8.234694e+00 |
| **max_bids_in_auct** | 1.982120e+02 | 1.024173e+03 |
| **max_bids_per_device** | 1.977450e+02 | 6.244694e+02 |
| **max_bids_per_device_per_auct** | 3.669694e+00 | 1.741291e+01 |
| **percent_fast_bids** | 4.855995e-03 | 3.977513e-02 |
| **bids_per_auct** | 6.343722e+00 | 2.428501e+01 |
| **bids_per_device** | 1.216968e+01 | 1.054413e+02 |
| **bids_per_url** | 1.538989e+01 | 1.226588e+02 |
| **device_per_auct** | 1.367550e+00 | 1.416368e+00 |
| **ip_per_ctry** | 1.650010e+01 | 8.161532e+01 |
| **percent_max_bids** | 4.250265e-01 | 3.512170e-01 |

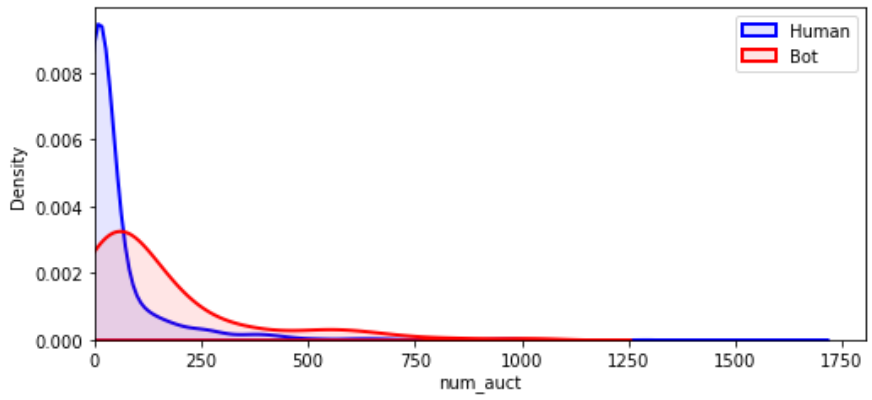
In [40]:
```
train_set.shape
```

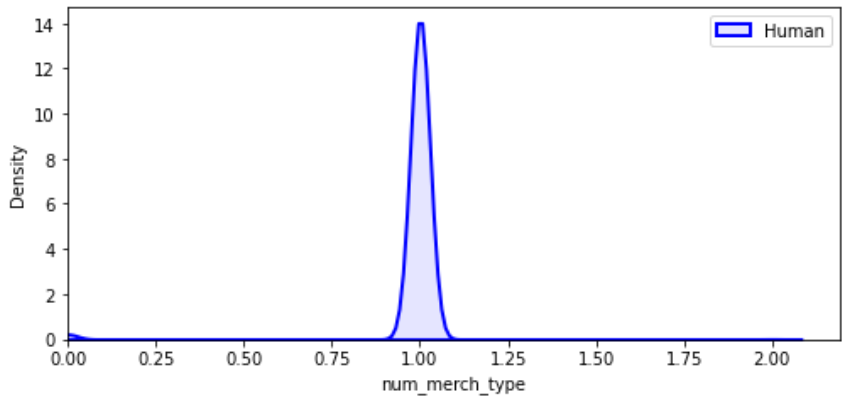
Out[40]:
```
(2008, 29)
```

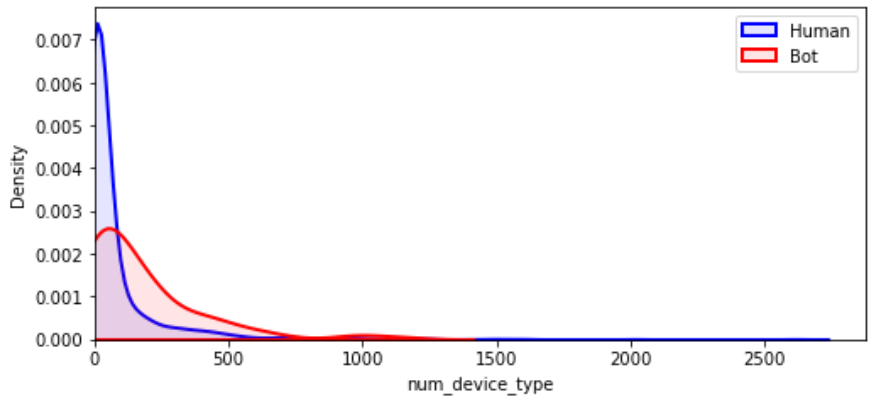
In [99]:
```
#plot density plot of all features to compare humans vs bots
import matplotlib.pyplot as plt
import seaborn as sns

features = train_set.columns.drop(['bidder_id', 'outcome'])

#denisty plot of bots vs humans
nrows = 14
ncols = 2
fig, axes = plt.subplots(nrows = nrows, ncols = ncols, figsize = (18, 60))
for i, column in enumerate(features):
    humans = sns.kdeplot(data = train_set[train_set['outcome'] == 0],
                x = column,
                ax = axes[i // ncols, i % ncols],
                color = 'blue',
                fill = True,
                alpha = 0.1,
                linewidth = 2,
                label = 'Human').set_xlim(left = 0)
    bots = sns.kdeplot(data = train_set[train_set['outcome'] == 1],
                x = column,
                ax = axes[i // ncols, i % ncols],
                color = 'red',
                fill = True,
                alpha = 0.1,
                linewidth = 2,
                label = 'Bot').set_xlim(left = 0)
    axes[i // ncols,i % ncols].legend()

plt.savefig('density.png', bbox_inches='tight')
plt.show()
```

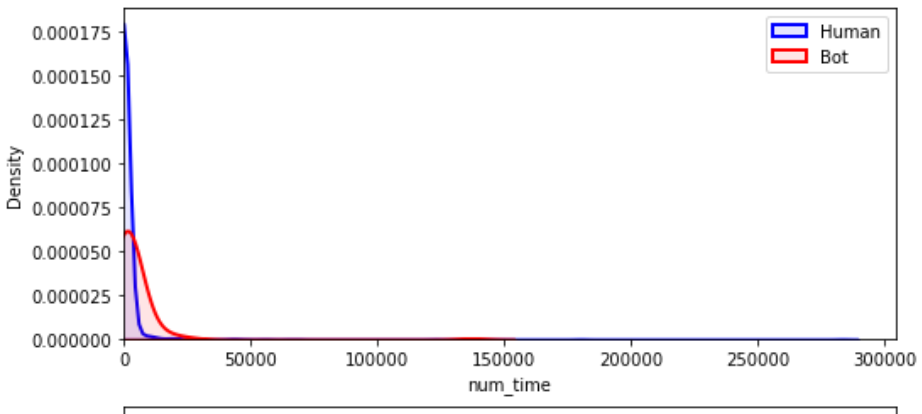
```
/Users/muthukumaran/opt/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py:316: UserWarning: Dataset has 0 variance; skipping density estimate. Pass `warn_singular=False` to disable this warning.
  warnings.warn(msg, UserWarning)
```

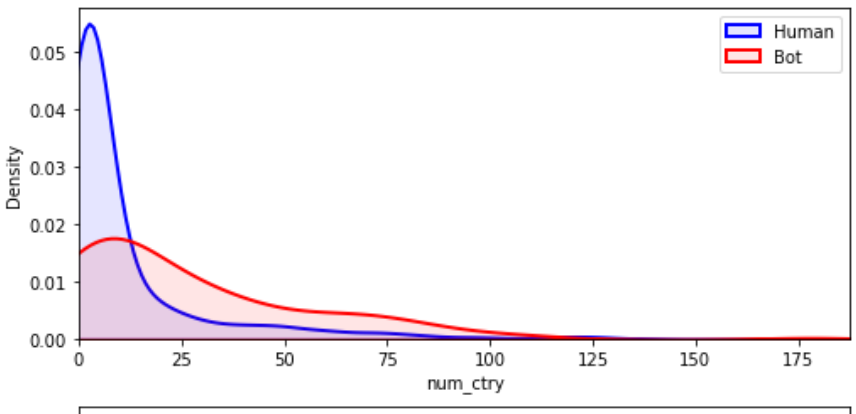

Human
Bot
Density
num_bids

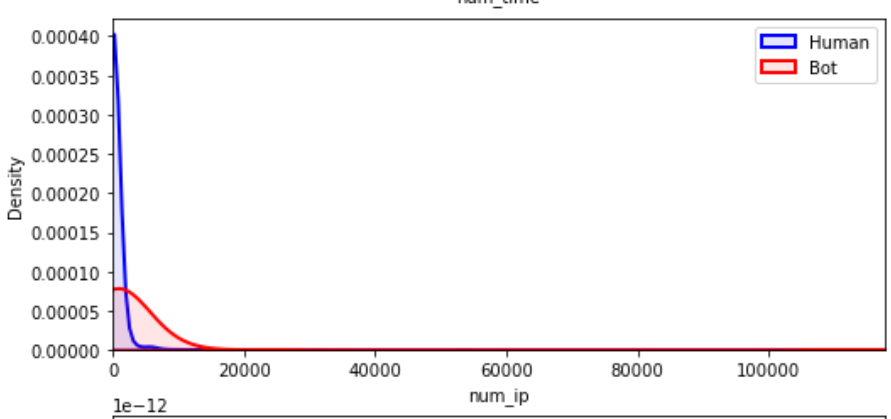

Human
Bot
Density
num_auct

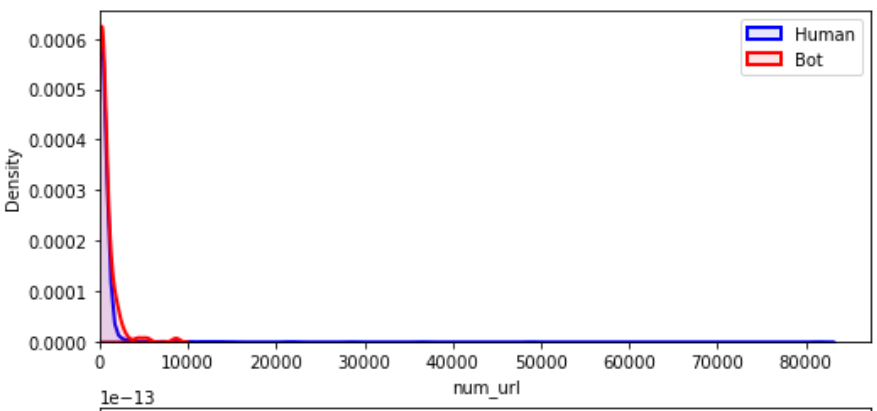

Human
Density
num_merch_type

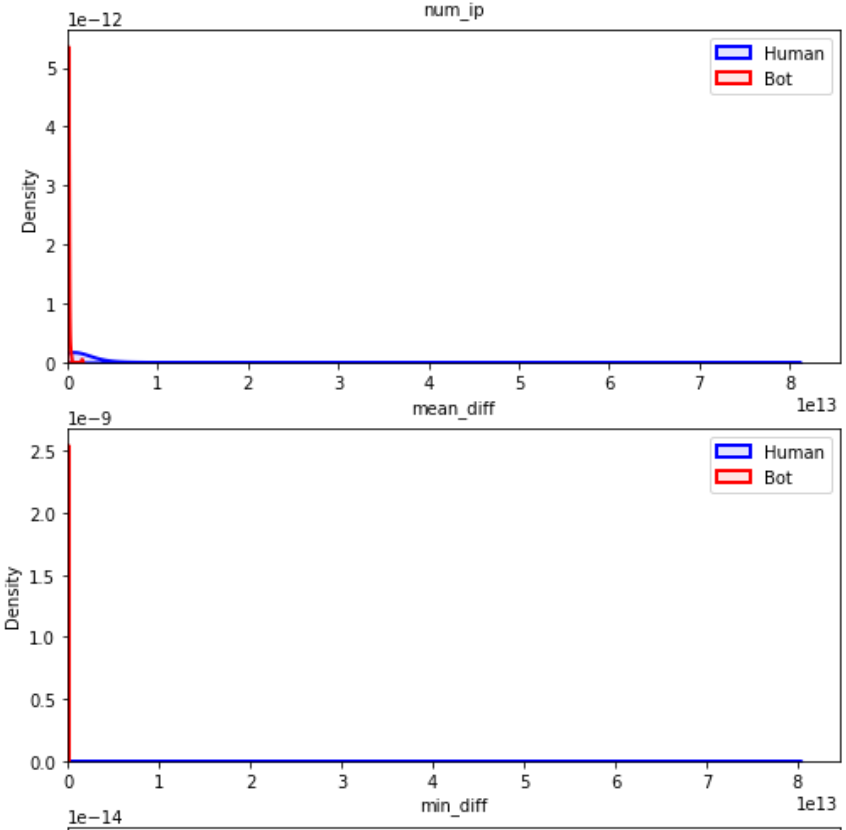

Human
Bot
Density
num_device_type

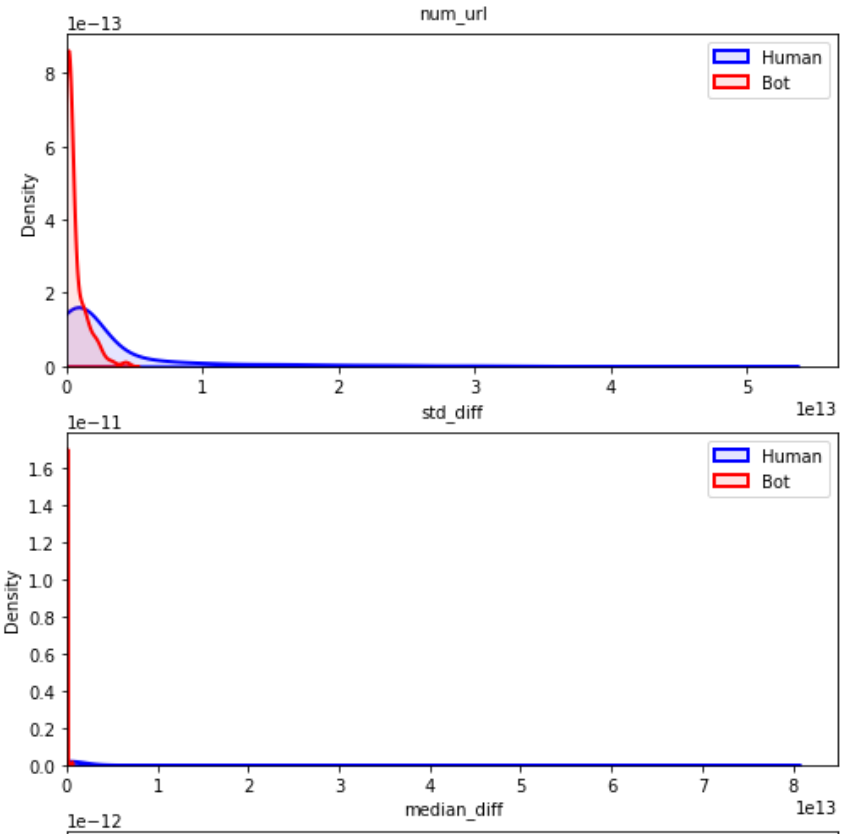

Human
Bot
Density
num_time


Human
Bot
Density
num_ctry


Human
Bot
Density
num_ip

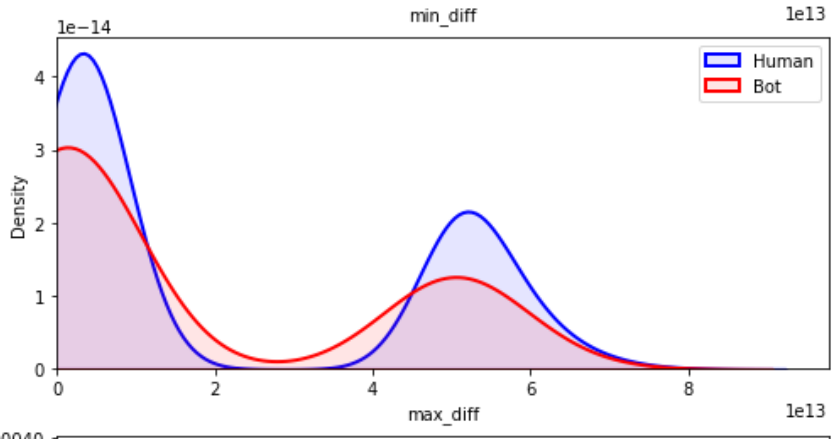

Human
Bot
Density
num_url
1e−13

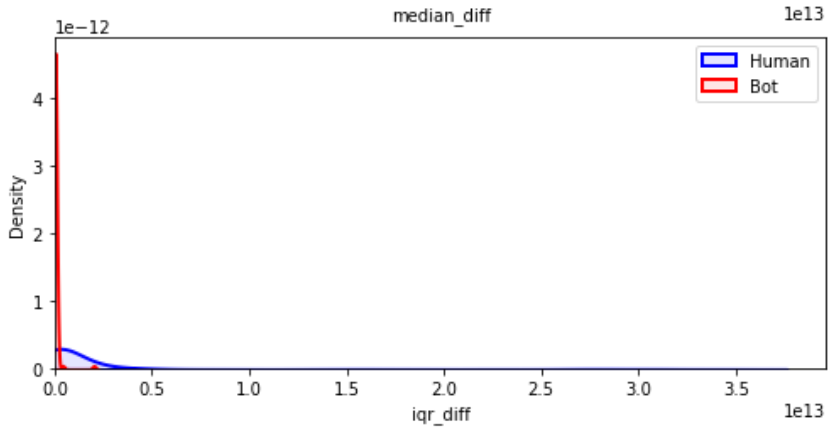

1e−12
Human
Bot
Density
mean_diff
1e−9
1e13


Human
Bot
Density
std_diff
1e−11
1e13


Human
Bot
Density
min_diff
1e13

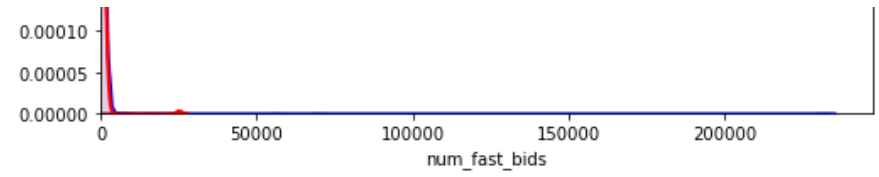

Human
Bot
Density
median_diff
1e13

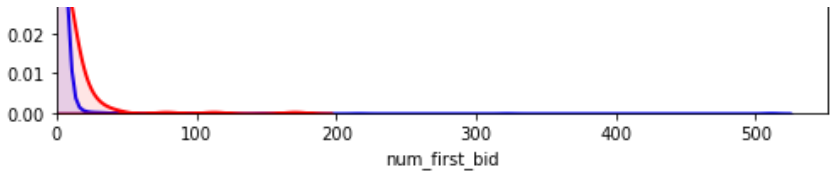

1e−14
Human
Bot
Density
max_diff
1e13

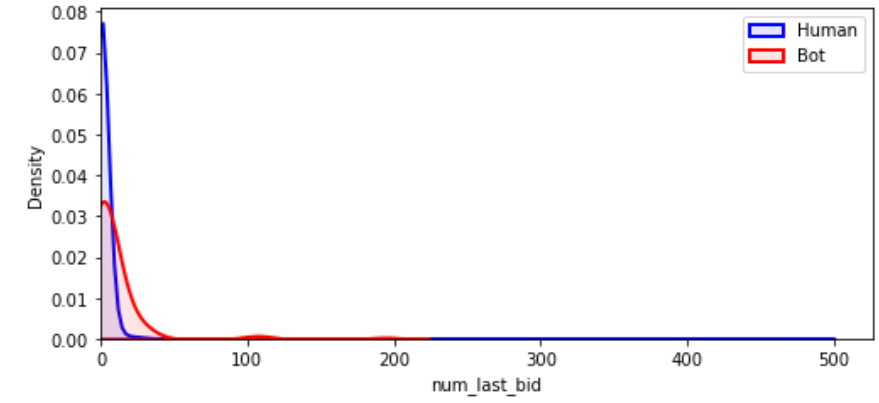

1e−12
Human
Bot
Density
iqr_diff
1e13

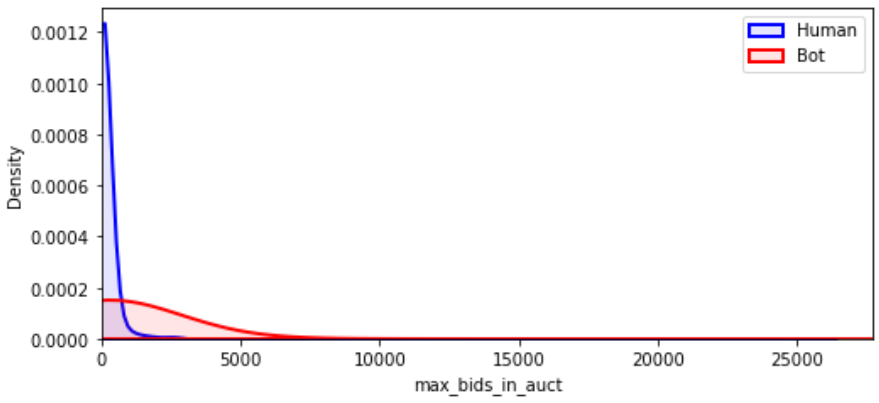

Human
Bot
Density

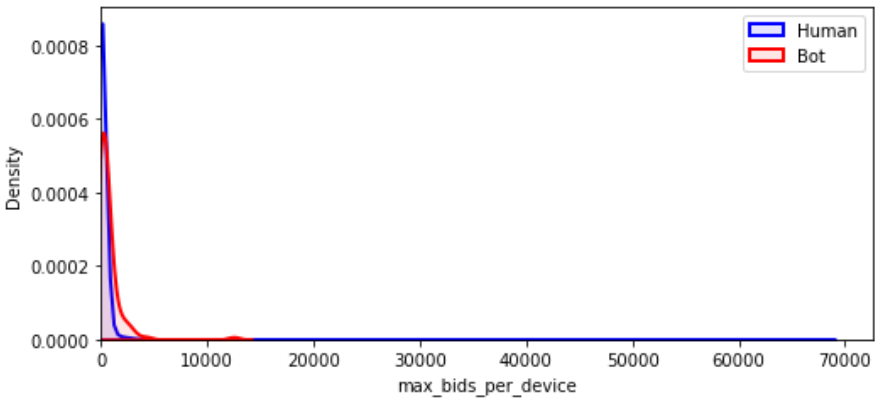

Human
Bot
Density

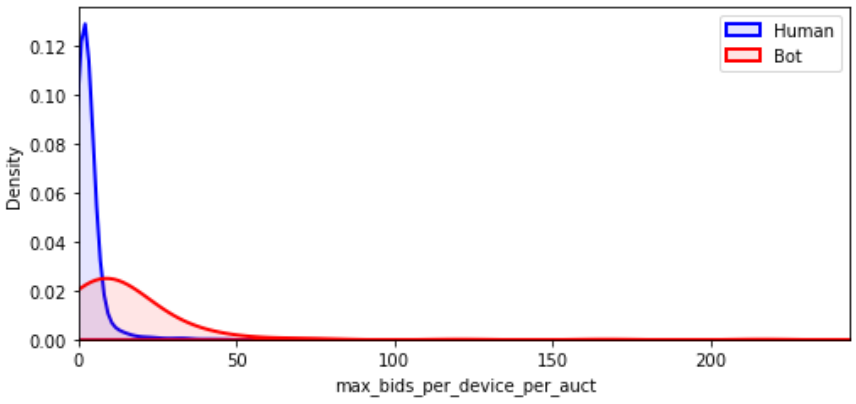


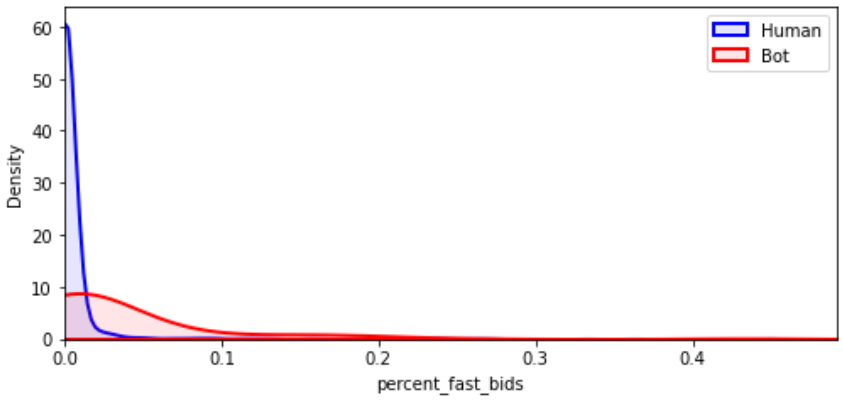



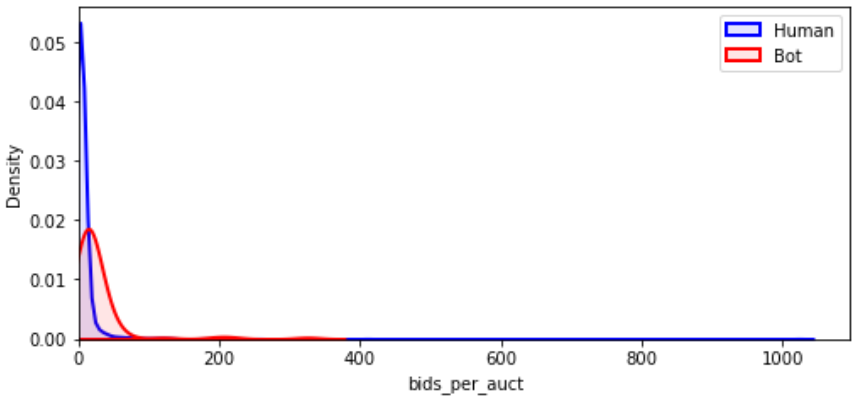



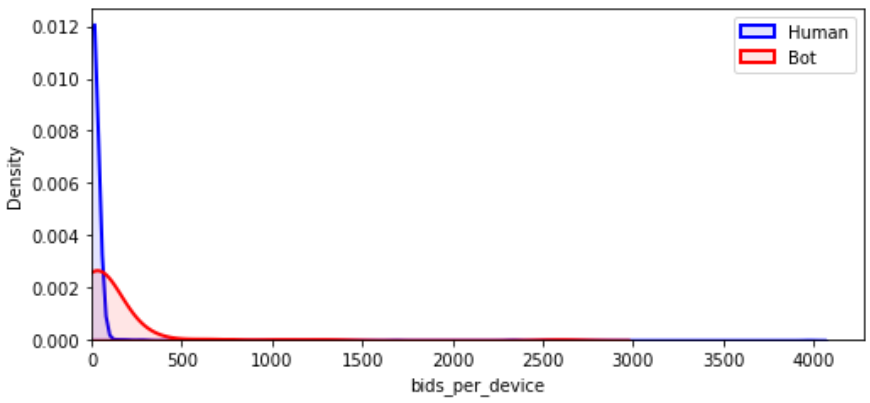



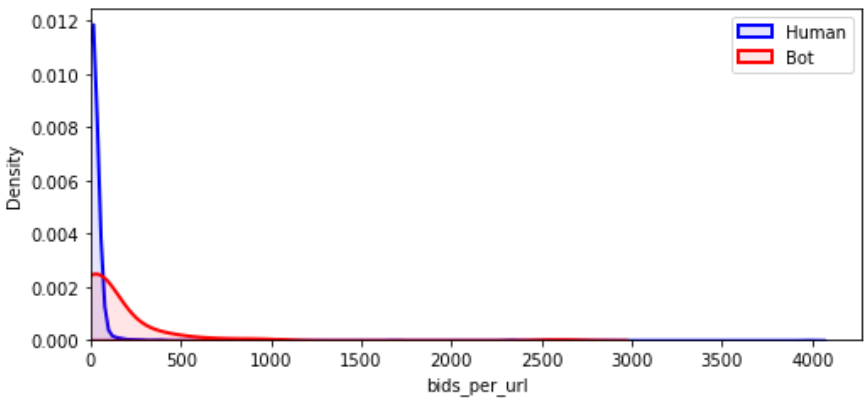



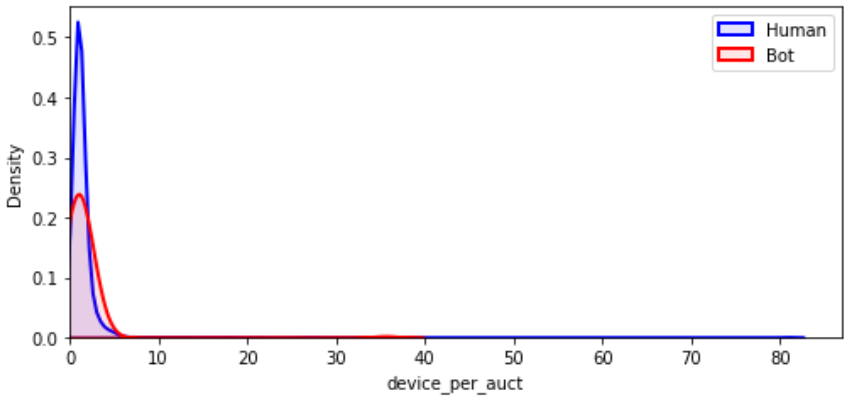



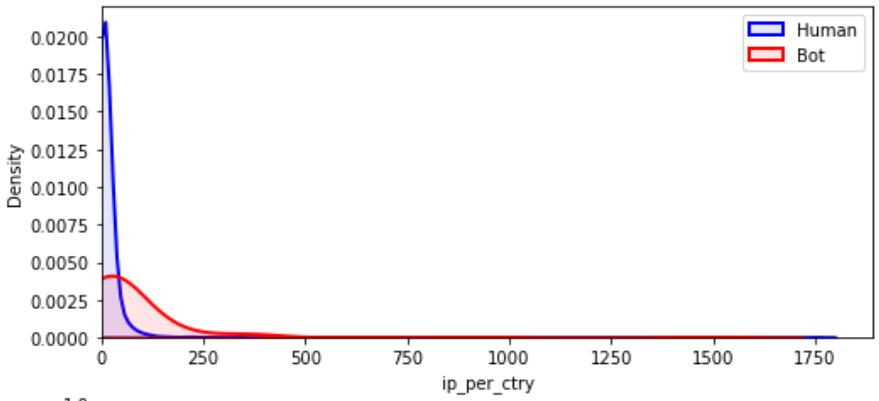



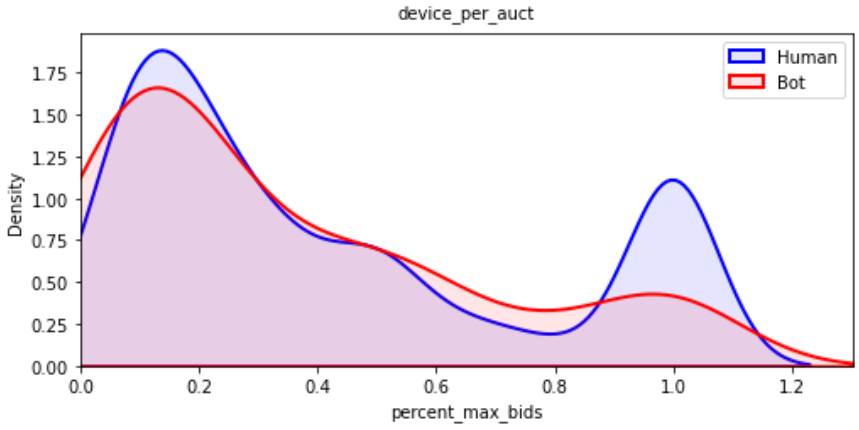



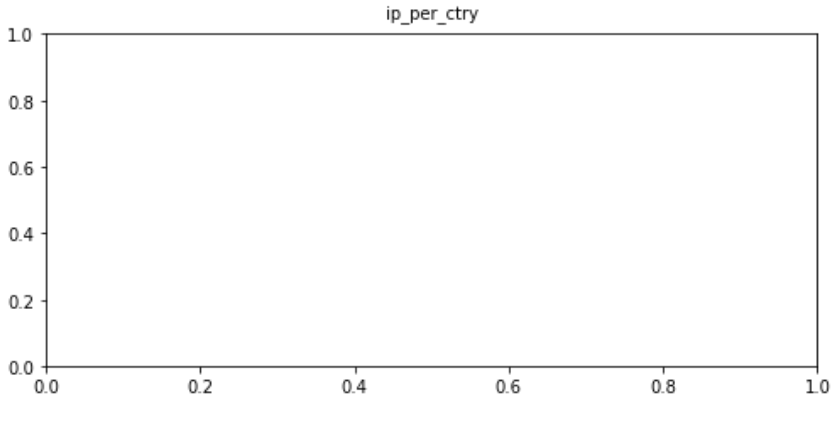



In [49]:
```
features = train_set.columns.drop(['bidder_id', 'outcome', 'num_merch_type',
                                   'num_fast_bids', 'max_bids_per_device', 'num_url'])
print(len(features))
features
X = train_set[features]
y = train_set['outcome']
```

```
23
```

In [53]:
```
#Cost benefit matrix -

cost_benefit_matrix = np.array([
```

```
    [0, -3],
    [-1, 0]])
```

In [54]:
```
from sklearn.metrics import confusion_matrix

def best_expected_value(y, y_scores):
    # Get thresholds
    prec, rec, thresholds = precision_recall_curve(y, y_scores)

    # Compute expected values at each threshold
    expected_values = []
    for t in thresholds:
        y_pred = y_scores > t # predictions at threshold
        conf_mat = confusion_matrix(y, y_pred, labels = [True, False], normalize = 'all')
        ev = (conf_mat * cost_benefit_matrix).sum()
        expected_values.append(ev)

    # Get maximum expected value
    return max(expected_values)
```

In [55]:
```
from sklearn.metrics import make_scorer

bev_score = make_scorer(best_expected_value, needs_threshold = True)
```

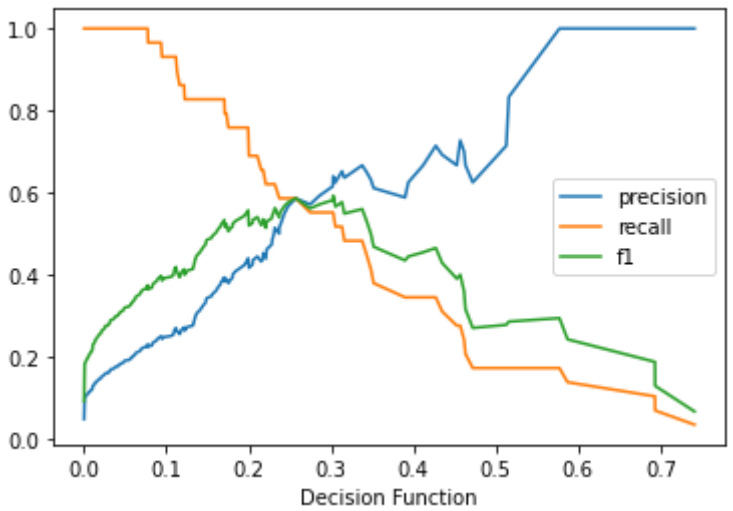
In [58]:
```
#Model random forest - hyper parameter tuning on training set
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
import xgboost as xgb
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_validate
from sklearn.metrics import precision_recall_curve

X = train_set[features]
y = train_set['outcome']

cv = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 42)

#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify = y )

#initialise model
rf_model = RandomForestClassifier(random_state = 42)

#initiate pipeline with scaling
pipe_rf = Pipeline([('scl', StandardScaler()),
                    ('RF',rf_model)])

#parameter range for gridsearchCV
param_range = [1, 2, 3, 4, 5, 6, 7, 8, 9]

#initialise parameter grid for gridsearch
rf_param_grid = [{'RF__min_samples_leaf': param_range,
                  'RF__max_depth': param_range,
                  'RF__min_samples_split': param_range[1:]}]

#initialise gridsearch pipeline
rf_grid_search = GridSearchCV(estimator=pipe_rf,
        param_grid=rf_param_grid,
        scoring=bev_score,
        cv=cv)

grids = [rf_grid_search]

for pipe in grids:
    pipe.fit(X,y)

grid_dict = {0: 'Random Forest'}

for i, model in enumerate(grids):
    print('{} Best EV: {}'.format(grid_dict[i],
    model.best_score_))
    print('{} Best Params: {}'.format(grid_dict[i], model.best_params_))
```

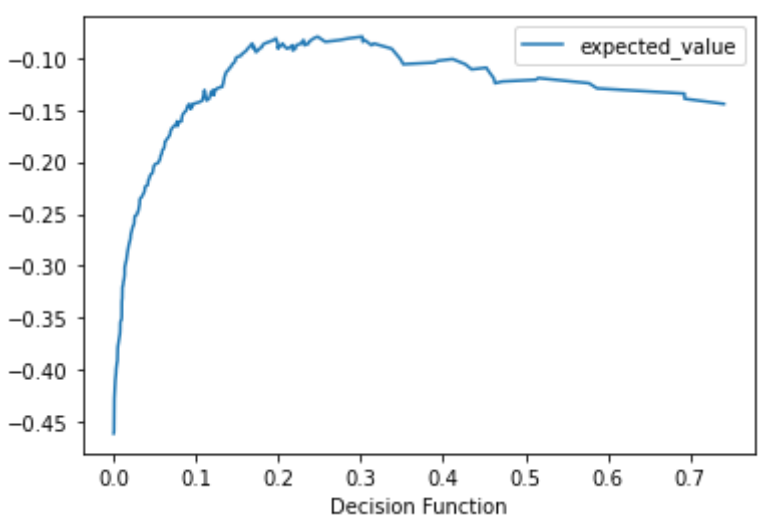
```
Random Forest Best EV: -0.07818141214128857
Random Forest Best Params: {'RF__max_depth': 8, 'RF__min_samples_leaf': 1, 'RF__min_samples_split': 4}
```

In [104…
```
#using train-test-split - Random Forest
from sklearn.metrics import accuracy_score, balanced_accuracy_score, precision_score, recall_score, f1_score, average_precision_s
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix


rf_model = RandomForestClassifier(random_state = 42, max_depth = 8, min_samples_leaf=1, min_samples_split=4)

pipe_rf = Pipeline([('scl', StandardScaler()),
```

```
                    ('RF',rf_model)])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify = y )

rf_model_fit = pipe_rf.fit(X_train, y_train)
y_proba = rf_model_fit.predict_proba(X_test)[:, 1]

y_pred_rf = y_proba > 0.247
prec, rec, thresholds = precision_recall_curve(y_test, y_proba, pos_label = True)

f1 = 2*prec*rec/(prec + rec)

# metrics vs thresholds plot
pr_df = pd.DataFrame(index = thresholds)
pr_df['precision'] = prec[:-1]
pr_df['recall'] = rec[:-1]
pr_df['f1'] = f1[:-1]
pr_df.plot()
plt.xlabel('Decision Function')

#ev vs thresholds plot
expected_values = []
for t in thresholds:
    # Compute predictions at that threshold
    y_pred = [True if score > t else False for score in y_proba]

    # Compute confusion matrix for those predictions
    mat = confusion_matrix(y_test, y_pred, labels = [True, False], normalize = 'all')

    # Calculate expected value
    expected_value = (mat * cost_benefit_matrix).sum()

    # Add to our list
    expected_values.append(expected_value)

ev_df = pd.DataFrame(index = thresholds)
ev_df['expected_value'] = expected_values[:]
ev_df.plot()
plt.xlabel('Decision Function')

#final metrics
rf_accuracy = accuracy_score(y_test, y_pred_rf)
rf_rocauc = roc_auc_score(y_test, y_proba)
rf_prcauc = average_precision_score(y_test, y_proba)
rf_recall = recall_score(y_test, y_pred_rf)
rf_precision = precision_score(y_test, y_pred_rf)
rf_f1 = f1_score(y_test, y_pred_rf)

t = thresholds[0]
ev = expected_values[0]
for thr, e_val in zip(thresholds,expected_values):
    if e_val > ev:
        ev = e_val
        t = thr

print(classification_report(y_test, y_pred_rf, labels = [True, False]))
print(f"Confusion matrix is:\n {confusion_matrix(y_test,y_pred_rf, labels = [True, False])}")

print('RF Accuracy train-test-split is:\n', rf_accuracy)
print('\nRF AUC_PRC train-test-split is:\n', rf_prcauc)
print('\nRF precision score train-test-split is:\n', rf_precision)
print('\nRF recall score train-test-split is:\n', rf_recall)
print('\nRF f1 score train-test-split is:\n', rf_f1)


print(f"Best Threshold:{t} with EV:{ev}.")
```

```
              precision    recall  f1-score   support

        True       0.57      0.59      0.58        29
       False       0.98      0.98      0.98       574

    accuracy                           0.96       603
   macro avg       0.77      0.78      0.78       603
weighted avg       0.96      0.96      0.96       603

Confusion matrix is:
 [[ 17  12]
 [ 13 561]]
RF Accuracy train-test-split is:
 0.9585406301824212

RF AUC_PRC train-test-split is:
 0.5960603492966177

RF precision score train-test-split is:
 0.5666666666666667

RF recall score train-test-split is:
 0.5862068965517241

RF f1 score train-test-split is:
 0.576271186440678
Best Threshold:0.24767363046025198 with EV:-0.07960199004975124.
```



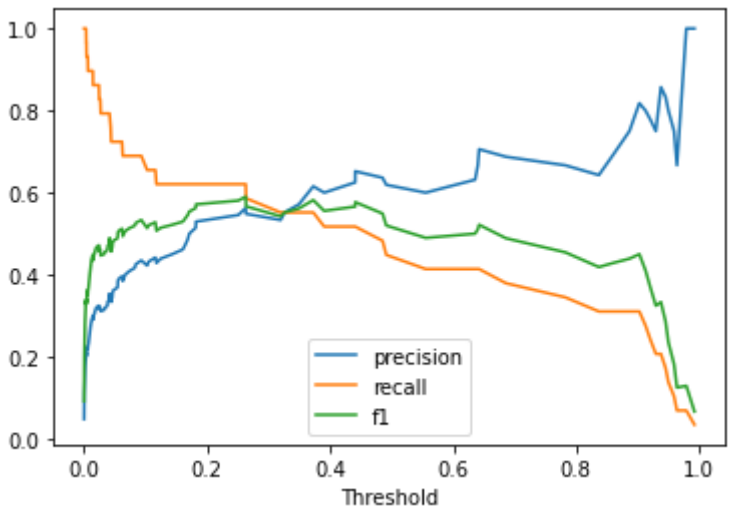

In [83]:
```
#Model XGBoost - hyper parameter tuning on training set

X = train_set[features]
y = train_set['outcome']
cv = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 42)
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify = y )

#initialise model
xgb_model = xgb.XGBClassifier(random_state=42)

#initiate pipeline with scaling
pipe_xgb = Pipeline([('scl', StandardScaler()),
                     ('XGB', xgb_model)])

#parameter range for gridsearchCV
param_range = [1, 2, 3, 4, 5, 6]
param_range_fl = [1.0, 0.5, 0.1]
n_estimators = [50,100,150]
learning_rates = [.1,.2,.3]

#initialise parameter grid for gridsearch
xgb_param_grid = [{'XGB__learning_rate': learning_rates,
                   'XGB__max_depth': param_range,
                   'XGB__min_child_weight': param_range[:2],
                   'XGB__subsample': param_range_fl,
                   'XGB__n_estimators': n_estimators}]

#initialise gridsearch pipeline
xgb_grid_search = GridSearchCV(estimator=pipe_xgb,
        param_grid=xgb_param_grid,
        scoring=bev_score,
        cv=cv)

grids = [xgb_grid_search]

for pipe in grids:
    pipe.fit(X,y)

grid_dict = {0: 'XGBoost'}

for i, model in enumerate(grids):
    print('{} Best EV: {}'.format(grid_dict[i],
    model.best_score_))
    print('{} Best Params: {}'.format(grid_dict[i], model.best_params_))
```

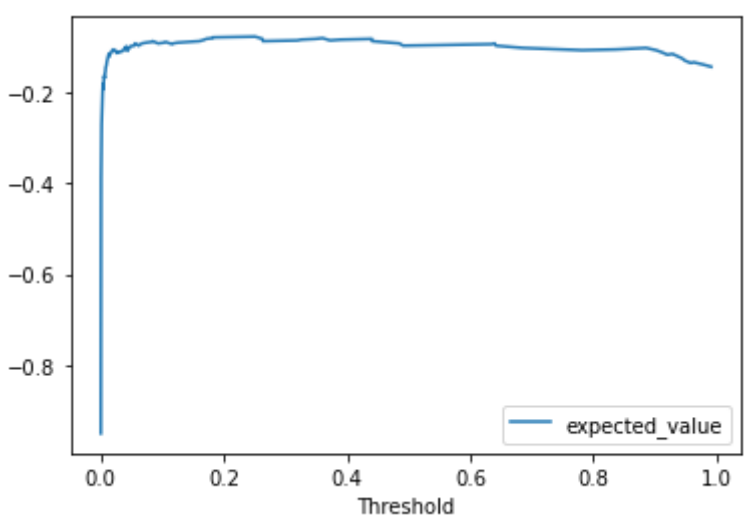
```
XGBoost Best EV: -0.08366645575116934
XGBoost Best Params: {'XGB__learning_rate': 0.3, 'XGB__max_depth': 6, 'XGB__min_child_weight': 1, 'XGB__n_estimators': 150, 'XGB
__subsample': 1.0}
```

In [105…
```
#using train-test-split - XGBoost

xgb_model = xgb.XGBClassifier(learning_rate=0.3, random_state=42, max_depth=6, min_child_weight=2,
                              n_estimators=150, subsample=1.0)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify = y )

pipe_xgb = Pipeline([('scl', StandardScaler()),
                     ('XGB', xgb_model)])
```

```python
xgb_model_fit = pipe_xgb.fit(X_train, y_train)
y_proba = xgb_model_fit.predict_proba(X_test)[:, 1]

y_pred_xgb = y_proba > 0.2507
prec, rec, thresholds = precision_recall_curve(y_test, y_proba, pos_label = True)


f1 = 2*prec*rec/(prec + rec)

# metrics vs thresholds plot
pr_df = pd.DataFrame(index = thresholds)
pr_df['precision'] = prec[:-1]
pr_df['recall'] = rec[:-1]
pr_df['f1'] = f1[:-1]
pr_df.plot()
plt.xlabel('Threshold')

#ev vs thresholds plot
expected_values = []
for t in thresholds:
    # Compute predictions at that threshold
    y_pred = [True if score > t else False for score in y_proba]

    # Compute confusion matrix for those predictions
    mat = confusion_matrix(y_test, y_pred, labels = [True, False], normalize = 'all')

    # Calculate expected value
    expected_value = (mat * cost_benefit_matrix).sum()

    # Add to our list
    expected_values.append(expected_value)

ev_df = pd.DataFrame(index = thresholds)
ev_df['expected_value'] = expected_values[:]
ev_df.plot()
plt.xlabel('Threshold')

#final metrics
xgb_accuracy = accuracy_score(y_test, y_pred_xgb)
xgb_rocauc = roc_auc_score(y_test, y_proba)
xgb_prcauc = average_precision_score(y_test, y_proba)
xgb_recall = recall_score(y_test, y_pred_xgb)
xgb_precision = precision_score(y_test, y_pred_xgb)
xgb_f1 = f1_score(y_test, y_pred_xgb)

t = thresholds[0]
ev = expected_values[0]
for thr, e_val in zip(thresholds,expected_values):
    if e_val > ev:
        ev = e_val
        t = thr

print(classification_report(y_test, y_pred_xgb, labels = [True, False]))
print(f"Confusion matrix is:\n {confusion_matrix(y_test,y_pred_xgb, labels = [True, False])}")

print('XGB Accuracy train-test-split is:\n', xgb_accuracy)
print('\nXGB AUC_PRC train-test-split is:\n', xgb_prcauc)
print('\nXGB precision score train-test-split is:\n', xgb_precision)
print('\nXGB recall score train-test-split is:\n', xgb_recall)
print('\nXGB f1 score train-test-split is:\n', xgb_f1)

print(f"Best Threshold:{t} with EV:{ev}.")
```

```
              precision    recall  f1-score   support

        True       0.55      0.62      0.58        29
       False       0.98      0.97      0.98       574

    accuracy                           0.96       603
   macro avg       0.76      0.80      0.78       603
weighted avg       0.96      0.96      0.96       603

Confusion matrix is:
 [[ 18  11]
 [ 15 559]]
XGB Accuracy train-test-split is:
 0.956882255389718

XGB AUC_PRC train-test-split is:
 0.5820530289311096

XGB precision score train-test-split is:
 0.5454545454545454

XGB recall score train-test-split is:
 0.6206896551724138

XGB f1 score train-test-split is:
 0.5806451612903226
Best Threshold:0.25072765350341797 with EV:-0.0779436152570481.
```




In [88]:
```
#Model LR - hyper parameter tuning on training set

X = train_set[features]
y = train_set['outcome']

cv = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 42)
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify = y )

#initialise model
lr_model = LogisticRegression(random_state=42)

#initiate pipeline with scaling
pipe_lr = Pipeline([('scl', StandardScaler()),
                    ('LR', lr_model)])

#parameter range for gridsearchCV
param_range_fl = [1.0, 0.75, 0.5, 0.25,  0.1]


#initialise parameter grid for gridsearch
lr_param_grid = [{'LR__penalty': ['l1','l2'],
                  'LR__C': param_range_fl}]

#initialise gridsearch pipeline
lr_grid_search = GridSearchCV(estimator=pipe_lr,
        param_grid=lr_param_grid,
        scoring=bev_score,
        cv=cv)

grids = [lr_grid_search]

for pipe in grids:
    pipe.fit(X,y)

grid_dict = {0: 'LogisticRegression'}

for i, model in enumerate(grids):
    print('{} Best EV: {}'.format(grid_dict[i],
    model.best_score_))
    print('{} Best Params: {}'.format(grid_dict[i], model.best_params_))
```

```
LogisticRegression Best EV: -0.11902085582064739
LogisticRegression Best Params: {'LR__C': 1.0, 'LR__penalty': 'l2'}
```

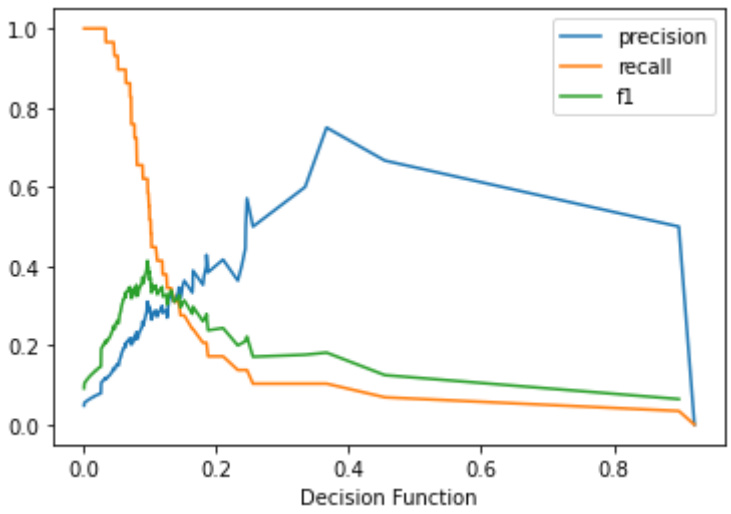
```
/Users/muthukumaran/opt/anaconda3/lib/python3.9/site-packages/sklearn/model_selection/_validation.py:378: FitFailedWarning:
25 fits failed out of a total of 50.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
25 fits failed with the following error:
Traceback (most recent call last):
  File "/Users/muthukumaran/opt/anaconda3/lib/python3.9/site-packages/sklearn/model_selection/_validation.py", line 686, in _fit
_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/Users/muthukumaran/opt/anaconda3/lib/python3.9/site-packages/sklearn/pipeline.py", line 382, in fit
    self._final_estimator.fit(Xt, y, **fit_params_last_step)
  File "/Users/muthukumaran/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py", line 1091, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/Users/muthukumaran/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py", line 61, in _check_sol
ver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/Users/muthukumaran/opt/anaconda3/lib/python3.9/site-packages/sklearn/model_selection/_search.py:953: UserWarning: One or more o
f the test scores are non-finite: [        nan -0.11902086         nan -0.12051463         nan -0.12001712
         nan -0.12300592         nan -0.12649223]
  warnings.warn(
```

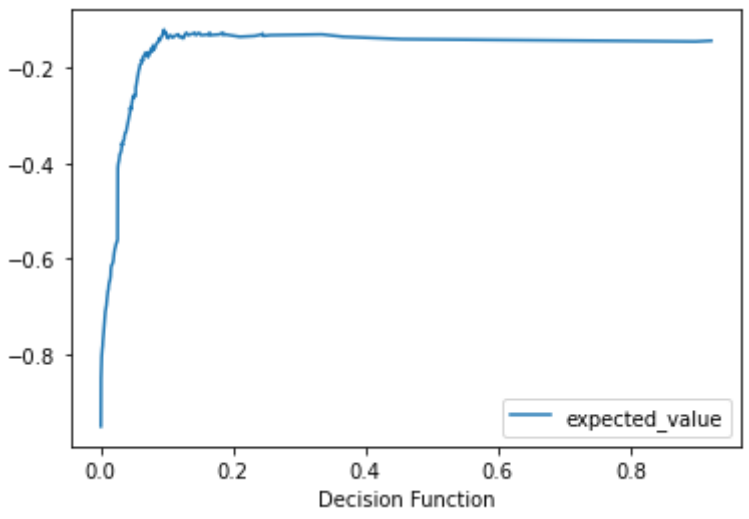
In [106…

```python
#using train-test-split - Logistic Regression

lr_model = LogisticRegression(random_state=42, penalty = 'l2', C= 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify = y )

pipe_lr = Pipeline([('scl', StandardScaler()),
                    ('LR', lr_model)])

lr_model_fit = pipe_lr.fit(X_train, y_train)
y_proba = lr_model_fit.predict_proba(X_test)[:, 1]

y_pred_lr = y_proba > 0.095
prec, rec, thresholds = precision_recall_curve(y_test, y_proba, pos_label = True)


f1 = 2*prec*rec/(prec + rec)

# metrics vs thresholds plot
pr_df = pd.DataFrame(index = thresholds)
pr_df['precision'] = prec[:-1]
pr_df['recall'] = rec[:-1]
pr_df['f1'] = f1[:-1]
pr_df.plot()
plt.xlabel('Decision Function')

#ev vs thresholds plot
expected_values = []
for t in thresholds:
    # Compute predictions at that threshold
    y_pred = [True if score > t else False for score in y_proba]

    # Compute confusion matrix for those predictions
    mat = confusion_matrix(y_test, y_pred, labels = [True, False], normalize = 'all')

    # Calculate expected value
    expected_value = (mat * cost_benefit_matrix).sum()

    # Add to our list
    expected_values.append(expected_value)

ev_df = pd.DataFrame(index = thresholds)
ev_df['expected_value'] = expected_values[:]
ev_df.plot()
plt.xlabel('Decision Function')

#final metrics
lr_accuracy = accuracy_score(y_test, y_pred_lr)
lr_rocauc = roc_auc_score(y_test, y_proba)
lr_prcauc = average_precision_score(y_test, y_proba)
lr_recall = recall_score(y_test, y_pred_lr)
lr_precision = precision_score(y_test, y_pred_lr)
lr_f1 = metrics.f1_score(y_test, y_pred_lr)

t = thresholds[0]
ev = expected_values[0]
for thr, e_val in zip(thresholds,expected_values):
    if e_val > ev:
        ev = e_val
        t = thr

print(classification_report(y_test, y_pred_lr, labels = [True, False]))
print(f"Confusion matrix is:\n {confusion_matrix(y_test,y_pred_lr, labels = [True, False])}")

print('LR Accuracy train-test-split is:\n', lr_accuracy)
```

```
print('\nLR AUC_PRC train-test-split is:\n', lr_prcauc)
print('\nLR precision score train-test-split is:\n', lr_precision)
print('\nLR recall score train-test-split is:\n', lr_recall)
print('\nLR f1 score train-test-split is:\n', lr_f1)

print(f"Best Threshold:{t} with EV:{ev}.")
```

```
/var/folders/cf/_53pz7w174sb1d98fsh_vdd40000gn/T/ipykernel_19335/2696529848.py:17: RuntimeWarning: invalid value encountered in
true_divide
  f1 = 2*prec*rec/(prec + rec)
              precision    recall  f1-score   support

        True       0.30      0.62      0.40        29
       False       0.98      0.93      0.95       574

    accuracy                           0.91       603
   macro avg       0.64      0.77      0.68       603
weighted avg       0.95      0.91      0.93       603

Confusion matrix is:
 [[ 18  11]
 [ 42 532]]
LR Accuracy train-test-split is:
 0.912106135986733

LR AUC_PRC train-test-split is:
 0.3215512923603904

LR precision score train-test-split is:
 0.3

LR recall score train-test-split is:
 0.6206896551724138

LR f1 score train-test-split is:
 0.40449438202247195
Best Threshold:0.09539732419856858 with EV:-0.12106135986733002.
```




In [93]:
```
#Model SVM - hyper parameter tuning on training set

X = train_set[features]
y = train_set['outcome']

#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify = y )


#initialise model
svm_model = SVC(random_state = 42)

#initiate pipeline with scaling
pipe_svm = Pipeline([('scl', StandardScaler()),
                     ('SVM', svm_model)])

#parameter range for gridsearchCV

param_range_fl = [5.0, 4.0, 3.0, 2.0,1.0, 0.5, 0.1]


#initialise parameter grid for gridsearch
svm_param_grid = [{'SVM__kernel': ['linear', 'rbf'],
```

```
                    'SVM__C': param_range_fl}]

#initialise gridsearch pipeline
svm_grid_search = GridSearchCV(estimator=pipe_svm,
        param_grid=svm_param_grid,
        scoring=bev_score,
        cv=cv)

grids = [svm_grid_search]

for pipe in grids:
    pipe.fit(X,y)

grid_dict = {0: 'SVM'}

for i, model in enumerate(grids):
    print('{} Best EV: {}'.format(grid_dict[i],
    model.best_score_))
    print('{} Best Params: {}'.format(grid_dict[i], model.best_params_))
```

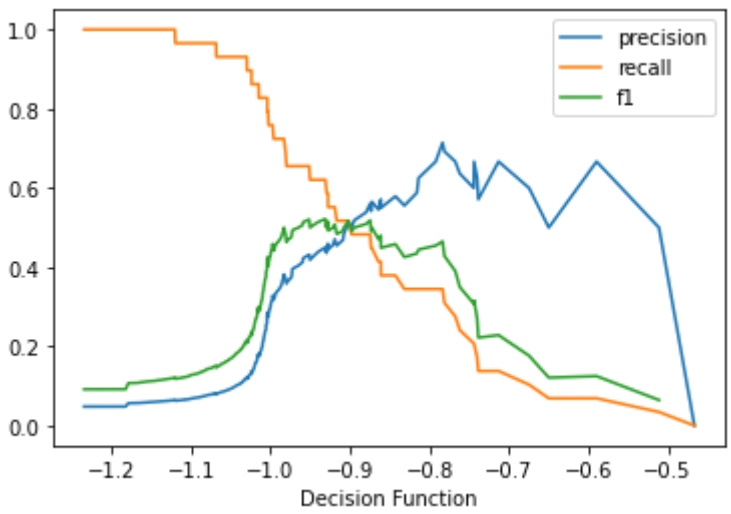
```
SVM Train AUC_PRC: -0.09909678539968488
SVM Best Params: {'SVM__C': 1.0, 'SVM__kernel': 'rbf'}
```

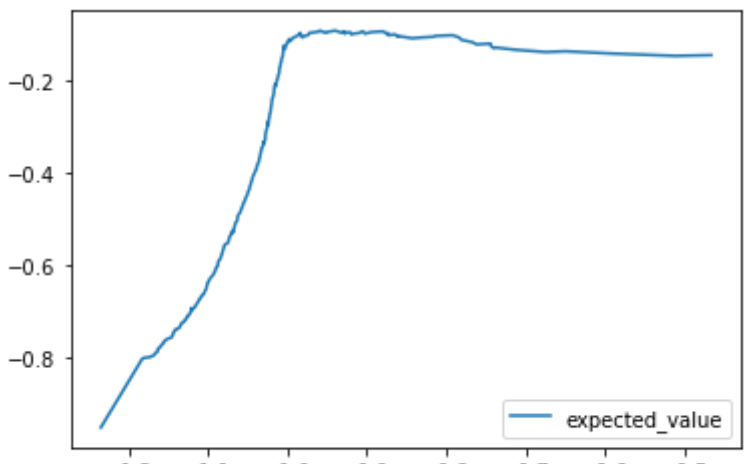
In [107…

```
#using train-test-split - SVM

svm_model = SVC(random_state=42, kernel = 'rbf', C = 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify = y )

pipe_svm = Pipeline([('scl', StandardScaler()),
                     ('SVM', svm_model)])

svm_model_fit = pipe_svm.fit(X_train, y_train)

y_proba = svm_model_fit.decision_function(X_test)

y_pred_svm = y_proba > -0.95

prec, rec, thresholds = precision_recall_curve(y_test, y_proba, pos_label = True)

f1 = 2*prec*rec/(prec + rec)

# metrics vs thresholds plot
pr_df = pd.DataFrame(index = thresholds)
pr_df['precision'] = prec[:-1]
pr_df['recall'] = rec[:-1]
pr_df['f1'] = f1[:-1]
pr_df.plot()
plt.xlabel('Decision Function')

#ev vs thresholds plot
expected_values = []
for t in thresholds:
    # Compute predictions at that threshold
    y_pred = [True if score > t else False for score in y_proba]

    # Compute confusion matrix for those predictions
    mat = confusion_matrix(y_test, y_pred, labels = [True, False], normalize = 'all')

    # Calculate expected value
    expected_value = (mat * cost_benefit_matrix).sum()

    # Add to our list
    expected_values.append(expected_value)

ev_df = pd.DataFrame(index = thresholds)
ev_df['expected_value'] = expected_values[:]
ev_df.plot()
plt.xlabel('Decision Function')


#final metrics
svm_accuracy = accuracy_score(y_test, y_pred_svm)
svm_rocauc = roc_auc_score(y_test, y_proba)
svm_prcauc = average_precision_score(y_test, y_proba)
svm_recall = recall_score(y_test, y_pred_svm)
svm_precision = precision_score(y_test, y_pred_svm)
svm_f1 = f1_score(y_test, y_pred_svm)

t = thresholds[0]
ev = expected_values[0]
for thr, e_val in zip(thresholds,expected_values):
    if e_val > ev:
        ev = e_val
        t = thr

print(classification_report(y_test, y_pred_svm, labels = [True, False]))
print(f"Confusion matrix is:\n {confusion_matrix(y_test,y_pred_svm, labels = [True, False])}")

print('SVM Accuracy train-test-split is:\n', svm_accuracy)
print('\nSVM AUC_PRC train-test-split is:\n', svm_prcauc)
print('\nSVM precision score train-test-split is:\n', svm_precision)
print('\nSVM recall score train-test-split is:\n', svm_recall)
```

```
print('\nSVM f1 score train-test-split is:\n', svm_f1)

print(f"Best Threshold:{t} with EV:{ev}.")
```

```
/var/folders/cf/_53pz7w174sb1d98fsh_vdd40000gn/T/ipykernel_19335/3066477396.py:18: RuntimeWarning: invalid value encountered in
true_divide
  f1 = 2*prec*rec/(prec + rec)
              precision    recall  f1-score   support

        True       0.43      0.62      0.51        29
       False       0.98      0.96      0.97       574

    accuracy                           0.94       603
   macro avg       0.70      0.79      0.74       603
weighted avg       0.95      0.94      0.95       603

Confusion matrix is:
 [[ 18  11]
 [ 24 550]]
SVM Accuracy train-test-split is:
 0.9419568822553898

SVM AUC_PRC train-test-split is:
 0.457797477231658

SVM precision score train-test-split is:
 0.42857142857142855

SVM recall score train-test-split is:
 0.6206896551724138

SVM f1 score train-test-split is:
 0.5070422535211268
Best Threshold:-0.9593349772397344 with EV:-0.0912106135986733.
```




In [ ]: