COPENHAGEN BUSINESS SCHOOL

BUSINESS DATA PROCESSING AND BUSINESS INTELLIGENCE

Data-driven wine procurement

Business analytics for finding and predicting great wines

Author

Christian Stolborg

Abstract

The global demand for wine is soaring with an expected increase in revenues of \$108b from 2020-2027, an increase of $\sim 33\%$ from today's market size of \$326b (Peterson, 2020). However, industry procurement is still largely based on conservative methods, often requiring the procurer to physically travel around wineries to taste their products. Such a process greatly limits any given procurers geographic field of expertise within wines and procurement is furthermore subject to personal opinions of the importer. In this paper a more data-driven approach is taken in an attempt to partially solve these problems. Based on the large online community Vivino.com we build a custom web-scraper in Selenium and download their vast wine database. The data is then loaded into our own database from which analytical procedures can easily be deployed, such as creating a large number of visualizations and training machine learning models. In an attempt to predict future highly rated vintages, the paper finds that tree-based ensemble models, such as the Random Forest and XGBoost, have the best out of sample performance with R^2 above 85%. The models are launched on `https://cbs-bdp-vivino.herokuapp.com/`, together with a Power Bi dashboard and source code, where the reader can use the ratings predictor on the fly. For source code see the Github at `https://github.com/Cstolborg/beyond-the-grape`.

Keywords: Business analytics, web scraping, Wine, Machine Learning, Predictive Modelling, No-Sql, Flask ,Wine Characteristics, Red Wine,.

Pages: 15 Characters w. spaces: 22,750 18. December 2020

Table of contents

1 Introduction

As a result of high global population growth, a rising middle-class in developing countries and generally a steady increase in consumer purchasing power, the demand for luxury goods are soaring (Peterson, 2019). Thus, the wine industry is expected to reach \$434b by 2027, up from \$326 in 2020 as a global distribution network connects consumers to wineries around the globe. However, the industry, which is still quite conservative, is largely dependent on traditional procurement methods. As such, many wine importers often have to physically visit wineries in their regions of choice, to personally taste wines and choose which ones to buy. Such a manual process naturally imposes certain limitations as to how many wines that the importer will have time to taste in a given time interval. This project seeks to alleviate these problems together with *Beyond the Grape*, a newly founded company trying to create a predictive tool for better assessment of future great wines.

The paper will focus on wines from the Vivino.com database. The purpose is to use the massive amounts of wines from the site to generate new data-driven insights and test whether machine learning models can be used to predict vintage ratings. Initially, a custom web-scraper is built since Vivino does not offer an open API. It is built in python and uses *Selenium*, *BeautifulSoup4* and *Requests* to parse their site. The data is initially stored in a No-Sql database in local JSON files after which it is transformed into a relational structure. It is preprocessed and cleaned for missing values, errors and other inconsistencies, as a means of preparing it for exploratory data analysis in Python & Power BI as well as to be used in predictive regression tasks with various machine learning models. The data is found to be non-linear and as such, an XGBoost has the best out of sample performance. The XGBoost model is saved in a joblib file and deployed to the web using Python's Flask package and Heroku as hosting. The site also hosts Power BI dashboards as well as a trial API where the reader can get a glimpse of the raw data structure. The reader is strongly suggested to check it out at `https://cbs-bdp-vivino.herokuapp.com/`.

Section 2 gives and in-depth explanation of the data collection and preprocessing strategy and in section 3 the data is visualized generating further insights. In section 4 regressive machine learning models are trained and deployed and in section 5 the conclusion is presented.

2 Data

In this section it is outlined how the data is collected, stored and used for analysis. The process is illustrated in the flow chart in Fig. 2.0.1. Initially, the Vivino website is web-scraped using Python and the packages Selenium, BeautifulSoup4 and Requests. The data is then stored in a No-Sql database ensuring flexibility and handling the horizontal structure of the data. In preparing the data for analysis, relations are constructed and the data is transformed and loaded into an SQL database. Then preprocessing is performed to account for missing values and errors and the data is outputted in a format that is ready for analytics. The data is then visualized in Power BI and parallel to that machine learning models are trained. Finally, the applications are launched to the web.

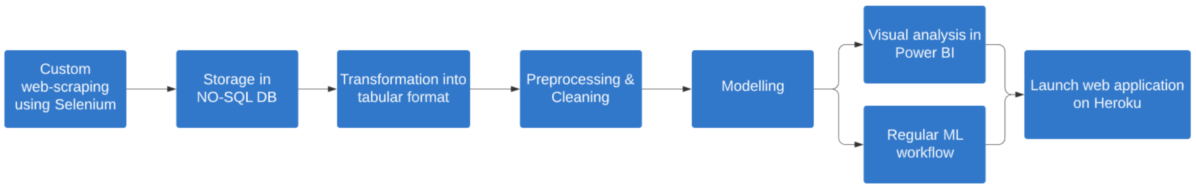

Custom web-scraping using Selenium
Storage in NO-SQL DB
Transformation into tabular format
Preprocessing & Cleaning
Modelling
Visual analysis in Power BI
Regular ML workflow
Launch web application on Heroku

Figure 2.0.1: Process Diagram. Since we don't have control over how the Vivino database evolves the data is intially stored in No-Sql and then transformed into relational format. This ensures potential future errors in the data fetch will arise internally in the system, rather than externally.

2.1 Data Collection

As mentioned, the analysis will be based off the Vivino.com website. Vivino is both an online wine marketplace as well as a community for wine enthusiasts. It is most readily comparable to a social network, in which the users can rate vintages from 1-5 stars, write reviews and follow each others profiles. As a result, Vivino hosts an array of information about its' vintages such as price, country of production, used grapes etc. but also a large amount of subjective attributes such as average ratings, average pct. sweetness/boldness in taste and user reviews. These attributes are measured as the average of all user contributed values.

Unfortunately, Vivino does not have an open API and therefore a custom crawler is developed in Python to acquire their data. The crawler ran from 31-11-2020 to 12-12-2020. Since some attributes has the need for horizontal scalability and, more importantly, because we don't control how the Vivino database will evolve in the future, the data is initially stored as JSON files. See Appendix A or the online Api made for the project[1] for an example. This will provide for a more flexible structure that can easily handle changes in the underlying attributes. Thus, the No-Sql structure likely prevents future crashes in the web-scraper. As a result, futures errors are more likely happen further into the data processing pipeline, but they will then be internal errors that are more easily fixed, rather than external errors that potentially can set the scraping significantly back in time.

As the data is highly hierarchical it is stored in 7 different JSON files, with appropriate keys to connect all files together. Concretely, each dataset contains inputs with its own key as well as a foreign key to the level one up in the hierarchy. The crawler yields data sets concerning vintages (an instance of a wine), wines, wineries, regions and coun-

[1] https://cbs-bdp-vivino.herokuapp.com/api/wines

tries. Additionally, the last two data sets concern information about reviews and users. The vintages dataset display information about each vintage such as price, rating and year of production. It is the product the users on Vivino.com can actually purchase. The wine dataset then holds aggregate information about vintages that has the same wine name, e.g. which grapes were used, how are the taste features such as level of sweetness/boldness etc. The wineries data contain information about each winery such as the winery's average rating, its geographical location, number of times it has been reviewed and so on. Regions and country datasets are primarily geographical information, even though it also holds information such as most popular grapes and the amount of users per region. Reviews dataset is simply information on each review and the users dataset contains a users number of followers, number of reviews given and more.

To limit the use of memory, the scraper divides files into temporary and permanent ones. The scraper is using basic *Python* dictionaries to store information when scraping, which for every n^{th} iteration saves the data into the temporary folders and resets the dictionaries. Therefore, for a single of the 7 different levels, there can be many temporary files, but only one permanent file. For every $n + j^{th}$ iteration, all the temporary files are compiled together into the permanent directory and the files here are thus updated. This step also handles potential duplicate values by not updating already existing index values in the permanent files. Once this is done, the temporary files are deleted and the process is continued until the scraper is finished. Apart from saving memory, the process also protects the permanent database from crashes in the scraper, since intermediate values are frequently saved and logged.

2.2 Tabularizing the data using ER modelling

The No-Sql storage made sense in the context of web-scraping as the high degree of flexibility results in a lower probability of errors caused by external factors. However, for analysis purposes it is not suitable and all data described above are transformed into a tabular format. The transformation is implemented by using the entity relational model in which each level in the data hierarchy represents an entity. As such, each entity will contain information related to the level of the hierarchy it represents.

Other models were also considered such as more traditional data warehousing techniques like the star or snowflake schemas. However, these approaches were not chosen even though they work very well with highly hierarchical data. The reason for this is grounded in the fact that due to the web-scraping dependency, at no point in time will it be certain that the full Vivino database is stored. As a result, when computing important aggregated metrics such as average winery ratings this would in some cases be based on an incomplete amount of ratings, making it deviate from its true value. However, Vivino.com offers these aggregated metrics themselves meaning the analysis will still be able to incorporate their true values rather than an approximation. Thus, the other approaches were not taken since some metrics cannot be aggregated from the lowest hierarchy and upwards. The relations are shown in Fig. 2.2.1.

From the entities shown in Fig. 2.2.1, the data can be easily tabulated from the 7 tables shown, into two tables that will be used for analysis. Note, that whilst some of the JSON files only contain simple key:value pairs that are easily converted to tabular, others contain more complex relationships with several layers. An example of the No-Sql storage with deeper horizontal levels is shown in Appendix A. Attributes with non-ordered categories like *foods* are one-hot encoded to account for the presence or non-presence of e.g. beef, chicken etc., and attributes *taste structure* and *flavor* are simply expanded into several columns thus unfolding their inherent key:value pairs.

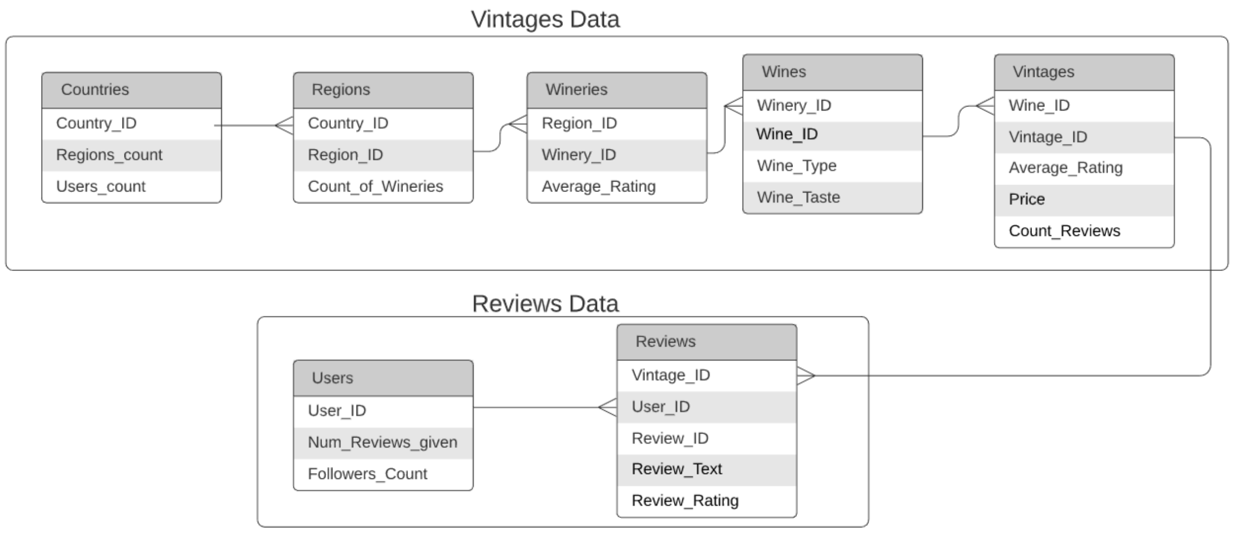


Figure 2.2.1: Data structure. The 7 data sets are combined into two, one that is vintage-centric and one that is review centric. Lines show cardinality. This is a non-exhaustive list of attributes for each data set.

In conclusion, the data has been transformed into a tabular format, from which relations are easily created between different entities. In the coming sections, the data will be explained from two views, a vintage centric view, and a reviews centric view. It should be noted that aggregating the data into one rather than two sets were also entirely possible, however, this would necessarily have required each vintage feature to be duplicated n_j times, where n_j is the number of n reviews per vintage j, which in this particular case is undesirable. This will be further explained in the machine learning section.

2.3 Data Description at Column Level & Preprocessing

A basic description of a selected subset of attributes is given in Table 2.3.1 along with descriptive statistics. This is merely an extract as there are 55 features in total. From features such as *price*, it is noted that the data is quite noisy with unrealistic prices at 0, as well as a standard deviation of ~ 8 times the mean. Furthermore, some variables contain a large number of NULL values. For now, the missing values are dropped reducing the vintages dataset to 48,452, however, many of these missing values were caused due to problems with the web crawler and future research should attempt fixing this issue and incorporate these into their analyses, since it is mostly attributable to the variable *Wine Type*. Furthermore, errors such as strings in integer and float variables are removed, e.g. *price: "350.0 kr."* is turned into 350.0. Filtering is applied to *year* and *price*, removing vintages produced earlier than 1980 and only including prices from 45DKK to 6,000DKK, since wines outside this range is bought at too low frequencies and are not of interest in the particular business case. Additionally, the top 1 and bottom 1 percentiles are removed to account for outliers. This threshold is set very conservatively as to not risk removing extreme values as these, contrary to outliers, are expected to be predictable. For further discussions on extreme values versus outliers see Nystrup (2018). All these implementations are performed in the Python library Pandas[2]. It should be mentioned that the reviews dataset is not used for further analysis in this paper, due to page limitations. However, a full NLP analysis on reviews is performed

[2]For source code see the self-contained notebook in the associated Github.

and can be seen in the self-contained notebooks on the GitHub.

Once transformations are performed the data is uploaded to an SQL server using the command line functionality. Storage data types follow standard conventions. DECIMAL(p,s) is used for decimal values with an appropriate precision and scale parameter. This is preferred as opposed to the FLOAT data type since the FLOAT type in certain scenarios return an approximation rather than the actual values. Most DECIMAL(p,s) features are set with $p = 38$ and $s = 20$ which should be more than enough for the current dataset. If values with more decimals than 20 should occur, it is preferred to be notified of this with an error as this may potentially require some changes to other parts of the system as well. INT(255) is used for all integer values, which should insure that the byte length is never exceeded. VARCHAR(n) is used for most categorical variables such as *year*, which we don't want to e.g. sum over. DATETIME is used for datetime variables where both the date and the time of day is stored.

Table 2.3.1: Descriptive statistics for a selected subset of attributes. There is a total of 55 different variables across all datasets. The reviews dataset is much larger since one vintage can have many reviews.

Feature	SQL Data Type	Count	Mean	Std	Min	Max	Description
Panel A: Vintages dataset							
Average rating	DECIMAL(38,20)	81,169	3.6	0.8	1.0	5.0	Avg. vintage rating from 1-5
Price (DKK)	DECIMAL(38,20)	81,169	324.0	2,724.2	0.0	287,626.5	Retail price in DKK
Rating count	INT(255)	81,169	840.9	2,549.7	0.0	122,412.0	No. of vintage ratings
Review count	INT(255)	81,169	188.7	372.5	0.0	22,669.0	No. of vintage reviews
Winery rating count	INT(255)	81,169	51,328.3	72,580.1	21.0	535,578.0	No. of winery ratings
Winery average rating	DECIMAL(38,20)	81,169	3.8	0.3	2.4	4.7	Avg. winery rating from 1-5
Wine Type	VARCHAR(255)	48,452	n.a.	n.a.	n.a.	n.a.	Red, white, sparkling etc.
Country	VARCHAR(255)	81,169	n.a.	n.a.	n.a.	n.a.	Country of production
Year	VARCHAR(255)	81,169	n.a.	n.a.	n.a.	n.a.	Year of production
Panel B: Reviews dataset							
Review	VARCHAR(8000)	281,781	n.a.	n.a.	n.a.	n.a.	Review of vintage
Rating	DECIMAL(38,20)	281,781	n.a.	n.a.	n.a.	n.a.	Rating $\in 1, 1.5...4.5, 5.0$
created_at	DATETIME	281,781	n.a.	n.a.	n.a.	n.a.	Datetime of review

Even though imputing strategies and feature engineering are not applied in this analysis, it could be a highly relevant strategy to apply in future uses with a heavier emphasis on building machine learning models. Additionally, the data is shown to be non-linear making Borup et. al's (2020) methods on linear variable selection as preparation step for non-linear learning potentially attractive preprocessing steps. In essence, it is a simple technique that entails tuning the C, or equivalently, the λ parameter in a LASSO regression to tune the number of non-zero $\hat{\beta}^{\lambda}$ variables, i.e. features in preparation for running other models such as a random forest. In python, this step can simply be added to the ML workflow using *sklearn's* Pipeline functionality and *GridSearchCV* to tune for the optimal number of features.

In conclusion, the section have provided an in-depth description of the data gathering process as well as how it is stored and preprocessed. Using a manual web-crawler the Vivino database was scraped and stored in a No-Sql format. From here, the data was transformed into relational format and subsequently cleaned for errors, missing values and other inconsistencies. A subset of features are then shown from the datasets along with a description of key relationships and their distributions. After preprocessing and slicing the final dataset ends up with 24,546 observations. In the next section, the data will be analyzed and further visualized in interactive dashboards and used in a range of machine learning models.

3 Business analytics for great wine selection

In the previous section, the methodology on gathering, storing and transforming the data were explained. In this section, we will take a deeper dive into the data. Concretely, the data will be visualized to get a better understanding of key relationships. This is done using a mix of Python and Power BI. Additionally, several machine learning models are trained with the purpose of predicting vintage ratings. As mentioned, wine importers usually have all intrinsic information about a vintage prior to purchasing it, except for its consumer rating. Using observable facts about the vintage, this section will thus attempt predicting how likely consumers are to enjoy a given vintage.

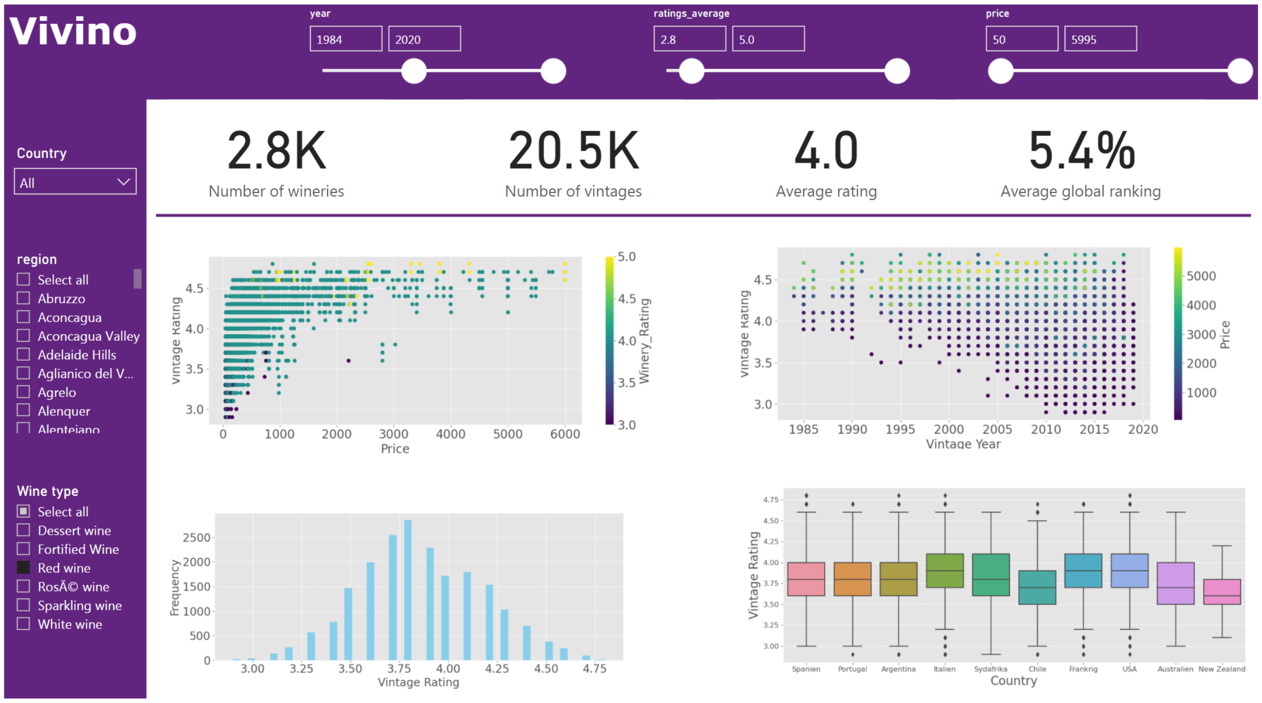
3.1 Visualization with Python and Power Bi

On the basis of the measures described in previous sections, the data is made ready for analytics. An explorative data analysis is performed with Python in Jupyter Notebook[3] as well as with Power BI, as it allows the user to interactively slice and subset many variables across the hierarchy to test for relationships. A major weakness to the dashboard setup is the difficulty in showing the interaction between many variables at once. Since this is highly dimensional data such a feature can be crucial in exploring the interactions between variables as well as their univariate distributions. A good implementation of this is created using the Python Pacakage *Pandas-Profiling*[4]. From the profiler interesting relationships appear to exist between variables such as *year*, *price*, *vintage rating* and *winery rating*. These are shown in a pairplot in Appendix B. From here, some preliminary conclusions can be made, for example, *price* and *year* seem to be conditionally linearly correlated with the target variable *vintage ratings* for certain subsets of the data. Likewise, *winery rating* appear almost unconditionally linearly correlated with *vintage ratings* suggesting wine importers should generally mostly consider high-quality wineries when choosing their vintages.

Based on the preliminary analysis done in Python, a Power Bi dashboard is built with an emphasis on *Vintage ratings*, *Price*, *Year* and *Winery Ratings*. Loading the data into Power BI is relatively simple since, as mentioned, all the preprocessing were completed using Python's pandas library which works very well with outputting csv and .xlsx files that integrates easily into Power Bi. In Power BI, it is merely required setting up the relations as shown in Fig. 2.2.1, and defining the data types on some variables. Based on that, the front-end graphs, slicers etc. are easily built. In Fig. 3.1.1, the dashboard is displayed. The primary purpose of the dashboard is to provide an interactive way of exploring key relationships in the data on various subsets. It is built with a number of slicers, allowing the user to subset the data on many dimensions. All the graphs are built using the internal Python editor with *Matplotlib* and *Seaborn*, which generally allows for more customization than the standard graphs in Power Bi. The primary drawdown of this approach is the loss of the ability to click on specific data-points in a graph and use these for slicing the remaining graphs. The only way to slice the these graphs are through external filters and slicers. However, as the most pressing issue in the particular business case revolves around building statistical models, the ability to slice all the way down to specific data points becomes less interesting as it takes away focus from the overall distributions. As a result, the usage

[3]For source code see the self-contained notebook in the associated Github.
[4]For the full profile in html format see associated GitHub.

of Python charts in Power BI is not seen to pose an issue for the time being.


Vivino

year 1984 2020

ratings_average 2.8 5.0

price 50 5995

Country All

region
- ☐ Select all
- ☐ Abruzzo
- ☐ Aconcagua
- ☐ Aconcagua Valley
- ☐ Adelaide Hills
- ☐ Aglianico del V...
- ☐ Agrelo
- ☐ Alenquer
- ☐ Alenteiano

Wine type
- ▣ Select all
- ☐ Dessert wine
- ☐ Fortified Wine
- ■ Red wine
- ☐ RosÃ© wine
- ☐ Sparkling wine
- ☐ White wine

2.8K Number of wineries

20.5K Number of vintages

4.0 Average rating

5.4% Average global ranking

Figure 3.1.1: Dashboard exhibit subsetted to show only red wines.

The dashboard features four different graphs along with 4 key indicators. All graphs are plotting vintage ratings. The top graphs show ratings' interaction with *vintage year*, *Winery rating* and *Price (DKK)*, whilst the bottom graphs are univariate and show various features of the underlying ratings' distribution. This ensures the target variable can always be seen in different dimensions. Interestingly, Fig. 3.1.1 show some non-linear patterns in the data, such as the top left graph, in which price and ratings seem to be highly linearly correlated at prices below 1500 DKK, whereas hereafter their relationship seems to fade out. It can also be concluded that vintages from before $\sim$ 2006 generally are associated with a higher rating than 3.5, with expensive vintages greatly increasing the probability of getting a rating above 4.0. Equivalently, the bottom left graph show that ratings centers around 3.75 and is slightly skewed. However, these conclusions are obviously only valid to the shown subset, and the distributions might easily change when performing further slicing on e.g. different types of wines. For example, it was found that white wines exhibit similar relationships, but the linearity between *price* and *vintage rating* turns nonlinear much sooner than reds, around 1000 DKK.

4 Machine learning models for better procurement strategies

For the particular business case, the dashboard and exploratory data analyses shall merely be seen as a means of providing an overview of the data. What is really interesting is the predictability of vintage ratings. As a result, several machine learning models are build and trained on the vintages dataset. The only additional modelling that is not yet described, is standard scaling defined as $z_i = \frac{x_i - \mu}{\sigma}$, where x_i is the i^{th} observation for feature x, mu and $sigma$ are the empirical means and standard deviations of feature x and z_i is the normalized score for observation x_i. This is applied to all continuous variables. The dataset is split in an 80:20 train/test set. As mentioned, the target variable is vintage ratings, and the independent variables are the remaining attributes in the vintages dataset, which were described in Table 2.3.1. When training the MSE will be used as the loss function, and in model evaluation we will further look into a risk-boundary, defined as

$$MSE = \frac{1}{n}\sum_{n=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{n=1}^{n}(\hat{\epsilon}_i)^2 \tag{4.0.1}$$

$$risk\_boundary = \frac{1}{N}\sum_{i=1}^{N}\mathbb{1}\{|\hat{\epsilon}_i| > 0.3\} \tag{4.0.2}$$

Where, $\hat{y}_i$ is the prediction for the i^{th} observation and y_i is the corresponding true value and the $\mathbf{1}\{\}$ is an indicator function (Géron, 2019). The risk-boundary, set to 0.3, is used to account for how many times the model makes extreme predictions, i.e. predictions with and error larger than 0.3. Since all errors are not born equal, the threshold is set as a 'maximum-pain' threshold of how wrong the model can be, for its predictions to still make sense from a purchasing perspective. This is an arbitrarily set number that has been determined together with the case company *Beyond the Grape*, who won't tolerate errors larger than this threshold.

Several machine learning models are trained. To ensure model diversity and thus maximize the probability that they are exposed to different types of errors, 3 types of models are trained, namely linear, tree-based and neural net. The linear and tree-based models are tuned using *sklearns GridSearchCV* functionality. The ANN architecture is iteratively tested through trial and error, starting with a 40x32x32x1, and ending with a 40x128x64x1. The ReLU activation function and adam optimizer is used[5]. A summary of their out-of-sample performance is shown in Table 4.0.1. It is clear that the tree-based ensemble models, Random Forest & XGBoost, by far has the best performance across all shown metrics. Unsurprisingly, as the data has some non-linear characteristics discussed previously, both linear models are outperformed. More surprisingly, the artificial neural network is also outperformed out-of-sample. This can be a result of either (1 poor model tuning and a need for a different architecture or 2) the ANN is overfitting the data, thus having a too high variance in and out-of-sample predictions.

Table 4.0.1: Out of sample performance of various models.

	Linear	Elastic Net	Decision Tree	Random Forest	XGBoost	ANN
MSE	0.035303	0.059330	0.034567	0.014411	0.013859	0.028673
R^2	63.3%	34.9%	69.3%	85.6%	86.0%	70.2%
risk_boundary	10.5%	21.4%	8.6%	2.3%	2.1%	7.7%

As the XGBoost slightly outperform the random forest we will focus on interpreting this model. From Table 4.0.1 it is clear that the XGBoost explains 86% of the dependent variable's (vintage rating) variance. More importantly, the risk-boundary parameter indicates that only 2.1% of predictions will be further off than 0.3 rating points. It can thus be concluded that the XGBoost's predictions are somewhat symmetrically centered around the actual out-of-sample values within the threshold of 0.3. This is largely confirmed in the scatter of predicted vs. actual values shown in Appendix C.

[5]For detailed explanations of model selection, cross validation, training, and evaluation see self-contained notebooks on the GitHub. Due to page limitations only the most important details are included in the paper.

A clear advantage of launching ML models, is that they can also be used to find new patterns in the data and inform us about these relationships. Since XGBoost is an ensemble model, evaluation can primarily be performed through feature importance's or better yet through SHAP values. Since feature importance has some major weaknesses such as inconsistency (Molnar, 2020), the SHAP values will be used here. SHAP values, which are based off coalition game theory, essentially computes, for each instance, the contribution of each feature to the prediction. In Fig. 4.0.1, the SHAP values for the 10 best predictors are shown in a violin plot. As can be seen, *price* and *year* are among the top 3, which serves as an additional entitlement for these variables being in the previously presented dashboard in Fig. 3.1.1. A high price generally seem to result in a higher model impact, although it can be noted that the majority of the mass has a shap value below 0.0. *Winery_ratings_average*, the second most important feature, also seem to positively correlate with the dependent variable, with a more gradual increase/decline right around 0.0. For *year*, the relationship is less straightforward, and due to the mix of red/blue values to either side, nothing can be inferred with respect to its overall correlation with the dependent variable. The only thing we can say from this plot, is that *year* serves as an important predictor but it is likely in some non-linear relationship. This is confirmed in Appendix D, in which vintage between 2005-2014 generally leads to a lower predicted rating, whereas vintages before and after this period leads to more positive performances.

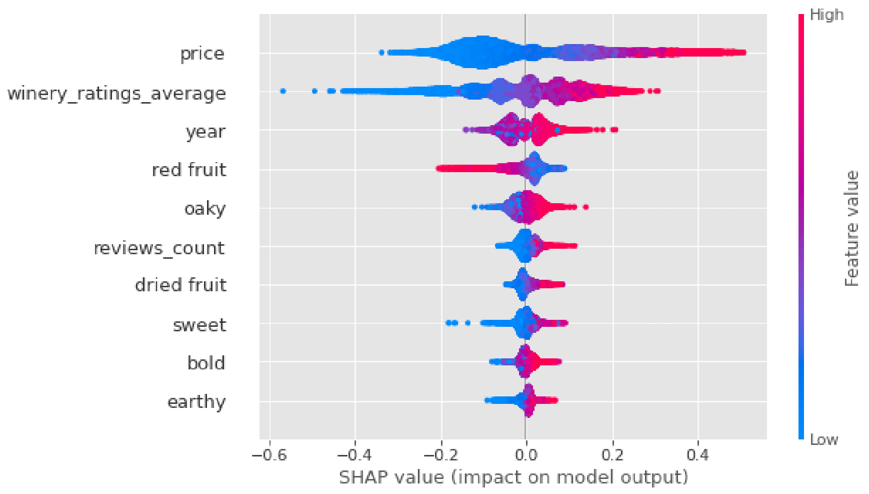


Figure 4.0.1: Violin plot of SHAP importances.

4.1 Recommended buying strategy

Using the visual analysis as well as the shap values from the previous sections, the following purchasing strategy is formulated for the case company *Beyond the Grape*. Appendix D contains additional partial SHAP importance plots, which will be the used as the primary source of this sections recommendations. As mentioned, vintages from 2005-2014 should be avoided altogether as implied by the shap values in Appendix D. Older vintages should primarily be bought from top-rated wineries, whilst new highly rated vintages can come from both medium and highly rated wineries. Furthermore, the procurer should, conditional on winery rating, seek inexpensive vintages as implicated by the SHAP values for winery rating, since the plot, somewhat counter intuitively, tends to rate cheap wines higher.

Additionally, Fig. 4.0.1 shows that vintages with little red fruit mostly generated a higher prediction and further, a buyer should be better off with vintages that is more *oaky* in flavour and contains more dried fruit contents. In summary, all these rules can generally be broken down to a general recommendation to look for great wineries first. It is much preferred to purchase cheap vintages from a great winery than the other way around. It is therefore recommended to look for wineries with already good brands.

4.2 Deployment with Flask to Heroku

The XGBoost model is deployed to a web-based application as a means of using the model in practice. The model is saved in joblib file and launched on `https://cbs-bdp-vivino.herokuapp.com/`. The site is created with Flask, a micro WSGI web framework written in Python. It is chosen due to its easy to use structure. It is deployed with Heroku. The site provides an easy to use tool, in which users can easily access the model and utilize it to select the best future wines on the fly. The site also contains the dashboards shown in the previous section, although Python charts are not yet supported in the free online version. Additionally, an excerpt of our data can be accessed through an Api endpoint at the site.

5 Conclusion

Utilizing the Vivino.com database, the paper has analyzed the possibility for a more data-driven approach to wine procurement. The analysis is carried out in the following 3 steps; Creating a data gathering & cleaning pipeline, generating insightful visualizations and training machine learning models. Data collection is performed by building a custom web crawler and storing the data in JSON files. The data is then transformed into tabular format and preprocessed for errors and missing values. Data analysis is carried out with Python and Power BI in which we find several interesting relations such as conditional linear dependence between *Vintage ratings* and *price*, *year*, *winery rating* respectively. Finally, several machine learning models are trained, and the XGBoost model is found to be the best in out of sample performance with an R^2 of 86% and being able to stay within risk-boundary, consisting of a max error of 0.3, up to 2.1% of the time. This concludes, that vintage ratings is somewhat predictable by using the regression models employed in the paper. Based on these analyses a recommended purchasing strategy is formulated, generally recommending any given importer to first buy wine from already known good wineries, and second, conditional on the winery rating, look for the relatively inexpensive wines. The models are launched on an online application using the Flask library, from which users can use the trained models in real time to make their own vintage rating predictions. Future research should focus on natural language processing on the reviews dataset, which were completely omitted form this paper. Initial such analyses are already ready on the GitHub in self-contained notebooks, which can be easily expanded[6]

[6]Due to limited pages it was excluded from this paper, even though it is found that vintage reviews holds a high potential for further finding extraordinary wines.

6 References

Ahlgren, P., Dahl, H., (2010). Hidden Markov models—detecting regimes in financial time series. *Nykredit Asset Management*, 1-9

Baum, L. E., Petrie, T., Richard E., (1966). Statistical Inference for Probabilistic Functions of Finite State Markov Chains, *The Annals of Mathematical Statistics*, 1 (1), 3-15

Bulla, J,, Mergner, S., Bulla, I., Sesboué, A. and Chesneau, C., (2011). Markov-switching asset allocation: Do profitable strategies exist?, *Journal of Asset Management*.

Cont, R. (2002). Empirical Properties of Asset Returns: Stylized Facts and Statistical Issues. *Quantitative Finance*, 1 (2), 223-236.

Dempster, A. P., Laird, N. M., and Rubin D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm, *Journal of the Royal Statistical Society*, 39 (1), 1-38.

Dungey, M., Martin, V. L., & Pagan, A. R. (2000). A multivariate latent factor decomposition of international bond yield spreads. *Journal of Applied Econometrics*, 15(6), 697-715.

Fama, Eugene F. and French, Kenneth R., (1989). Business conditions and expected returns on stocks and bonds, *Journal of Financial Economics*, 25, 23-49.

Frühwirth-Schnatter, S., 2006. Finite Mixture and Markov Switching Models. *Springer*.

Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. *O'Reilly Media*.

Nystrup, P., Lindström, E., Henrik Madsen, (2020). Learning hidden Markov models with persistent states by penalizing jumps.

Nystrup, P., Madsen, Henrik and Erik Lindstrom, 2009. *Long memory of financial time series and hidden Markov models with time-varying parameters.*. Journal of Forecasting.

Nystrup, P., (2014). Regime-Based Asset Allocation: Do profitable strategies exist?. *DTU Orbit*.

Quandt, R. E., (1958). The Estimation of the Parameters of a Linear Regression System Obeying Two Separate Regimes. *Journal of the American Statistical Association*, 53 (284), 873-880.

Quandt, R. E., (1972). A New Approach to Estimating Switching Regressions. *Journal of the American Statistical Association*, 67 (338), 306-310.

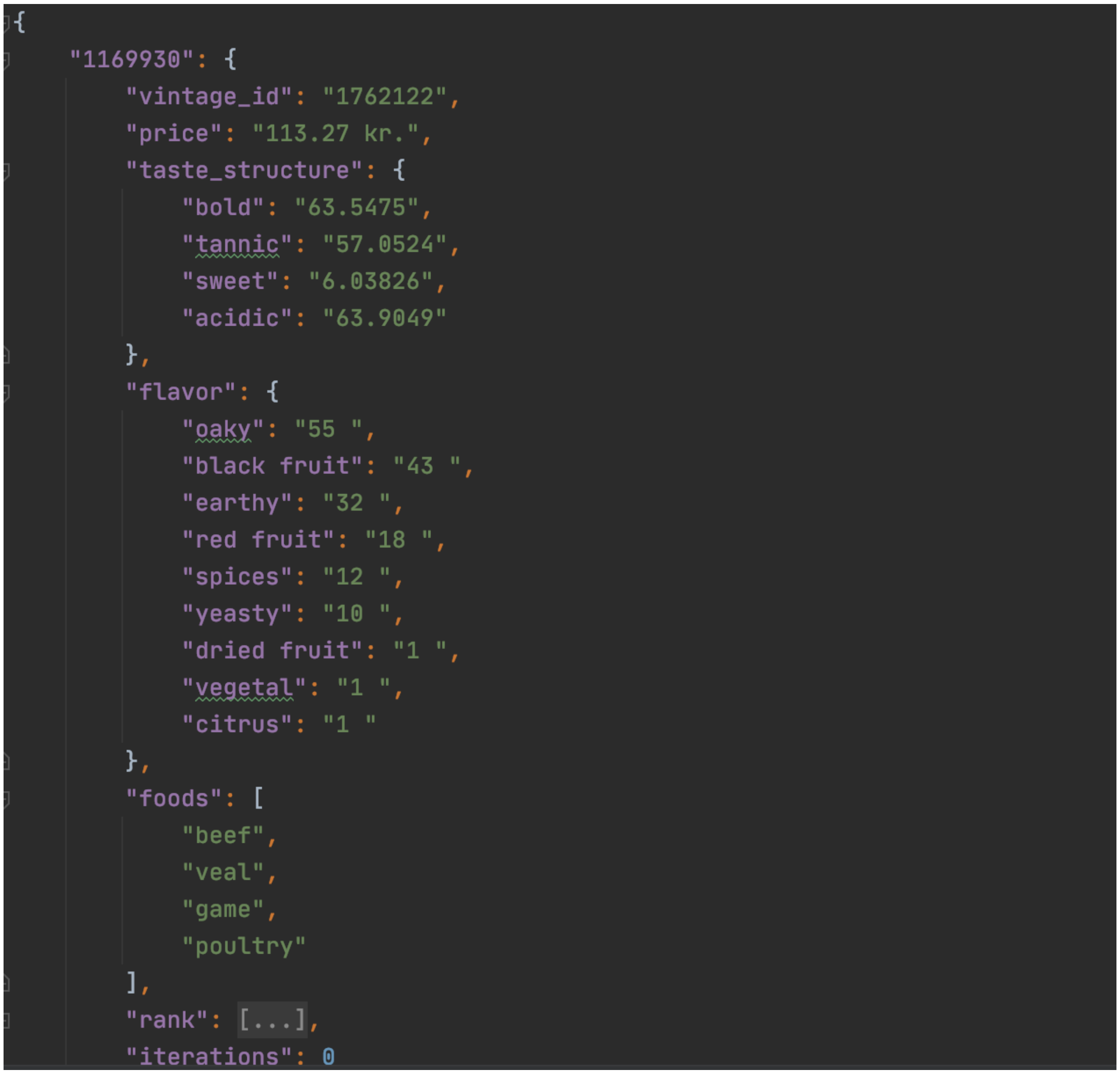
Quandt, Richard E., (1972). A New Approach to Estimating Switching Regressions. *Journal of the American Statistical Association*, 67 (338), 306-310.

Rabiner, L. R. (1989). A tutorial on Hidden Markov Models and selected applications in speech recognition, *Proceedings of the IEEE*, 77, 257–286.

Rydén, T., (2008). EM versus Markov chain Monte Carlo for estimation of hidden Markov models: A computational perspective. *Bayesian Analysis*, 3(4), 659–688.

7 Appendices

A Example of raw No-Sql format



```
{
    "1169930": {
        "vintage_id": "1762122",
        "price": "113.27 kr.",
        "taste_structure": {
            "bold": "63.5475",
            "tannic": "57.0524",
            "sweet": "6.03826",
            "acidic": "63.9049"
        },
        "flavor": {
            "oaky": "55 ",
            "black fruit": "43 ",
            "earthy": "32 ",
            "red fruit": "18 ",
            "spices": "12 ",
            "yeasty": "10 ",
            "dried fruit": "1 ",
            "vegetal": "1 ",
            "citrus": "1 "
        },
        "foods": [
            "beef",
            "veal",
            "game",
            "poultry"
        ],
        "rank": [...],
        "iterations": 0
```

Figure A.0.1: Wine_ID example. The primary key at the highest level refer to the wine_id, and the values are its features. The No-Sql is optimal for storage for the features *taste_structure, flavor & foods*, since we don't control how these attributes will evovle in the future.

B Pairplot

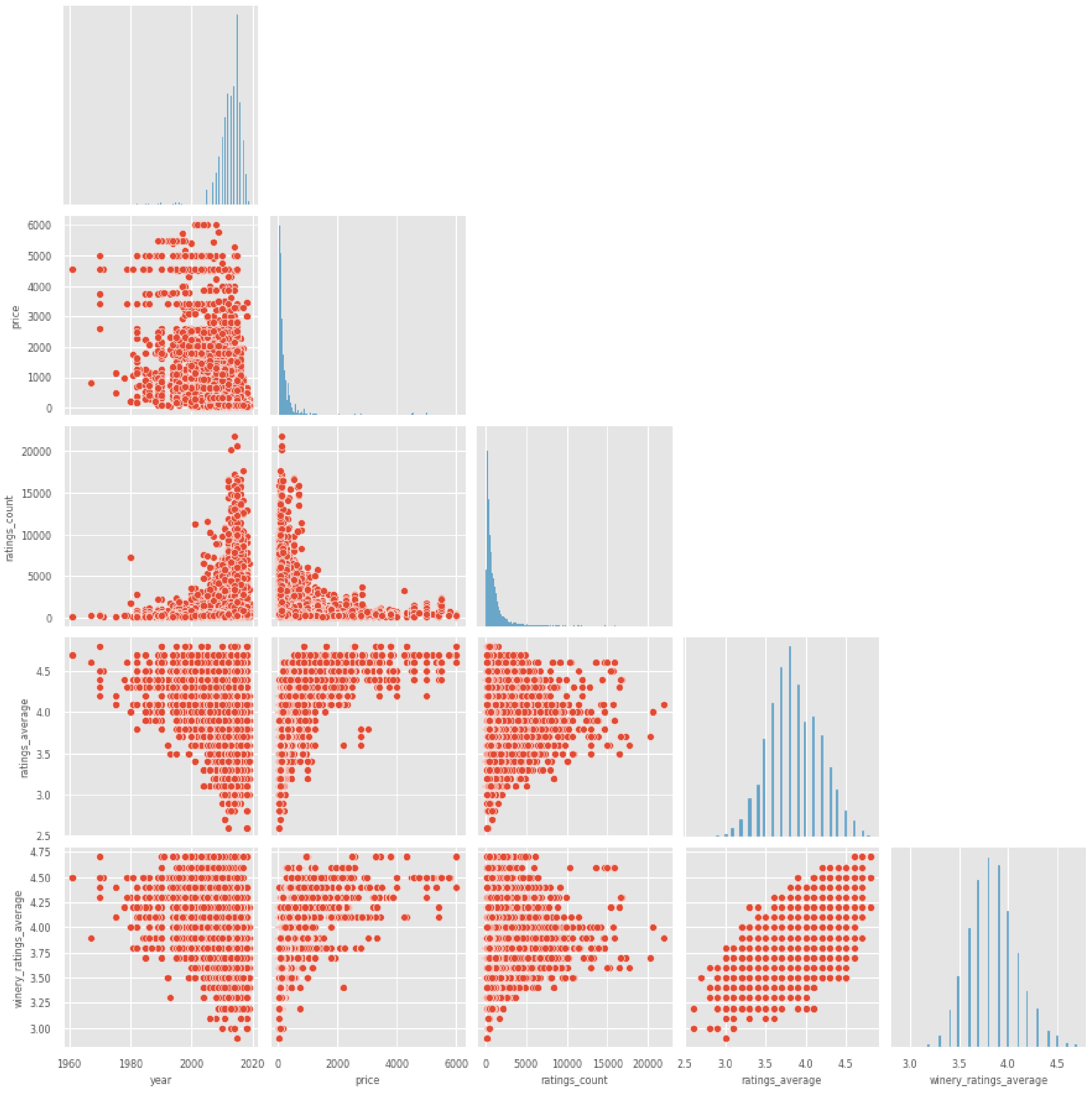


Figure B.0.1: Pairplot of selected variables.

C Random Forest & XGBoost predicted vs. actual values scatter

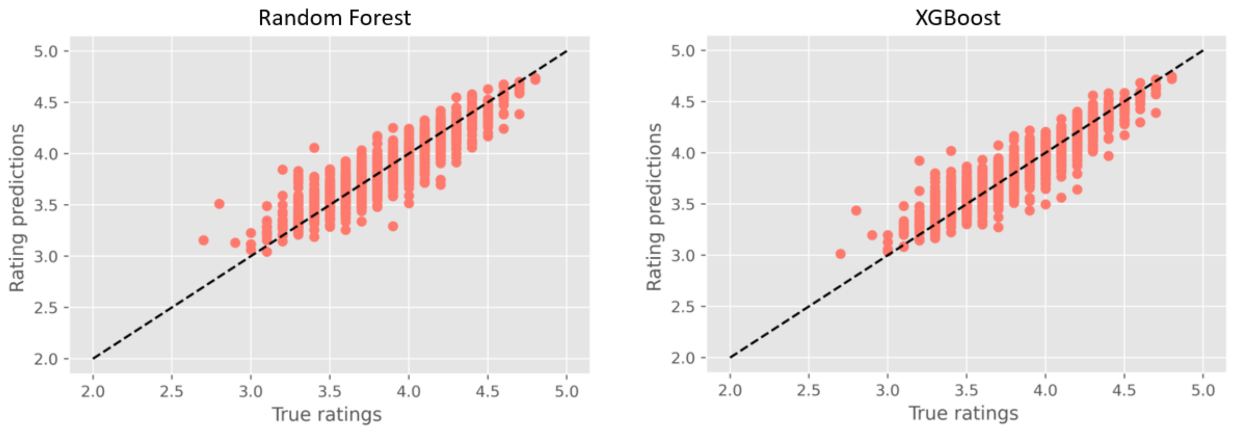


Figure C.0.1: Out of sample performance

D XGBoost SHAP partial feature importance plots

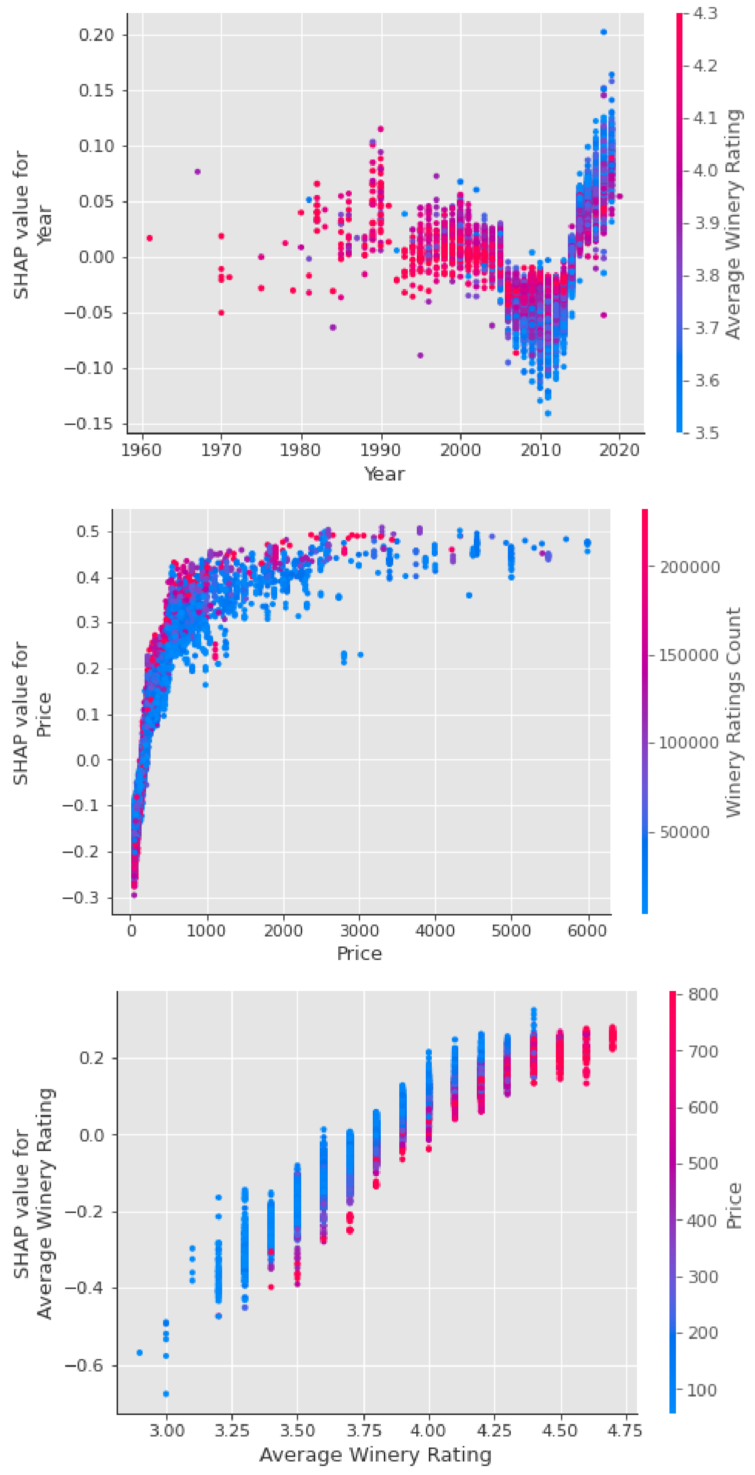


Figure D.0.1: SHAP partial importances plots.