WINE REVIEWS DATA ANALYSIS AND POINTS PREDICTION

By: Lawrence Adu-Gyamfi and Adrià Fenoy

Date: 28/10/2018

Table of contents

1. Introduction

The objective of this report is to detail the data analysis and machine learning techniques performed on a dataset. It details the steps followed to identify, pre-process and analyze the dataset in order to draw some insights from it. The results from the Exploratory Data Analysis have been used to identify relevant features to include in the building and training of the machine learning model. The model has been trained on a section of the data and later evaluated by testing it on the remaining test data. Some conclusions about the results of the model and its predictions are presented at the end of the report.

1.1. Scope of Study

The main purpose of the exercise is to build and train a machine learning model to predict the rating of a wine in terms of points. These points range from 1 to 100 which means the problem is of the regression kind. This training and prediction will be based on available information provided in the dataset in the form of different variables. The available variables are detailed in section 1.2.

1.2. Tools and Packages

The following packages have been used in the report for the studies:

- **Pandas and Numpy:** Provide the data structures framework for the data analysis.
- **Matplotlib and Seaborn:** For the data visualisation.
- **NLTK (Natural Language Toolkit):** Provides the stopwords and Stemming packages for manipulation of the text.
- **Tensorflow and keras:** For building and training the neural network model.
- **WordCloud:** For generating a wordcloud plot of the words in the descriptions

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
from wordcloud import WordCloud
```

1.3. Data Description

This data was obtain from the *Kaggle* dataset Wine Reviews. As described in the overview of this dataset, the data has been scraped from [WineEnthusiast] The dataset contains about 130.000 rows of wine reviews with the following featues:

- **Coutry:** The country where the wine comes from.
- **Description:** A summary provided of the wine provided by a sommelier describing the taste of the wine, smell, look, feel, etc.
- **Designation:** The vineyard where the grapes used to make the wine come from.
- **Points:** Points awarded by WineEnthusiast from 1 to 100. It is noted that they only post reviews for wines that score above 80 points.
- **Price:** Cost of a bottle of the wine.
- **Province:** Province or state the wine is from.
- **Region 1:** Wine growing area within province or state.
- **Region 2:** More specific growing area.
- **Taster name:** Sommelier who tasted and reviewed the wine.
- **Taster Twitter handle:** Twitter account of the sommelier.
- **Title:** The title summarising the wine review, often containing the vintage of the wine.
- **Variety:** Type of grapes used to make the wine.
- **Winery:** The winery that made the wine.

We import the data to a dataframe and show the first rows of the dataset to have a view with some examples of each feature and how the information is presented.

In [2]:

```
path = "~/Wines.csv"
wines = pd.read_csv(path, index_col = 0)
wines.head(4)
```

Out[2]:

	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_handle	title	
0	Italy	Aromas include tropical fruit, broom, brimston...	Vulkà Bianco	87	NaN	Sicily & Sardinia	Etna	NaN	Kerin O'Keefe	@kerinokeefe	Nicosia 2013 Vulkà Bianco (Etna)	
1	Portugal	This is ripe and fruity, a wine that is smooth...	Avidagos	87	15.0	Douro	NaN	NaN	Roger Voss	@vossroger	Quinta dos Avidagos 2011 Avidagos Red (Douro)	Port
2	US	Tart and snappy, the flavors of lime flesh and...	NaN	87	14.0	Oregon	Willamette Valley	Willamette Valley	Paul Gregutt	@paulgwine	Rainstorm 2013 Pinot Gris (Willamette Valley)	Pir
3	US	Pineapple rind, lemon pith and orange blossom ...	Reserve Late Harvest	87	13.0	Michigan	Lake Michigan Shore	NaN	Alexander Peartree	NaN	St. Julian 2013 Reserve Late Harvest Riesling ...	

To get a better idea of how we can deal with each feature, the table below presents a general overview in terms of statistics of the raw dataset.

In [3]:

```
wines.describe(include = 'all')
```

Out[3]:

	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_hand
count	129908	129971	92506	129971.000000	120975.000000	129908	108724	50511	103727	987
unique	43	119955	37979	NaN	NaN	425	1229	17	19	
top	US	This zesty red has pretty aromas that suggest ...	Reserve	NaN	NaN	California	Napa Valley	Central Coast	Roger Voss	@vossrog
freq	54504	3	2009	NaN	NaN	36247	4480	11065	25514	255
mean	NaN	NaN	NaN	88.447138	35.363389	NaN	NaN	NaN	NaN	Na
std	NaN	NaN	NaN	3.039730	41.022218	NaN	NaN	NaN	NaN	Na
min	NaN	NaN	NaN	80.000000	4.000000	NaN	NaN	NaN	NaN	Na
25%	NaN	NaN	NaN	86.000000	17.000000	NaN	NaN	NaN	NaN	Na
50%	NaN	NaN	NaN	88.000000	25.000000	NaN	NaN	NaN	NaN	Na
75%	NaN	NaN	NaN	91.000000	42.000000	NaN	NaN	NaN	NaN	Na
max	NaN	NaN	NaN	100.000000	3300.000000	NaN	NaN	NaN	NaN	Na

From the above statistics, based on the count and unique values, it is evident there are several missing information as well as repeitions in most of the variables. This is not a major concern at this stage as most of these variables are not considered for the training of the model based on the feature the model is being trained to predict. This is further expanded in the Data preprocessing section.

An interesting challenge presented, as evident from the data descriptions and statistics, is the datatype(mostly nominal) and uniqueness of the variables in the dataset. This eventually affected the method chosen for the model.

2.0. Data Pre-processing

The main idea of this study is to train the network to predict the points based on the description provided by the sommeliers after tasting the wine. The scenario assumed here is a typical blind tasting situation where the tasters are privy to no other information aside the probably the "general type of wine (the varietal and/or region) and the vintage. No information about the winery or the price of the wine is available to the taster while they are tasting" (https://www.winespectator.com/display/show/id/tasting-format)

This means, the description generated by the taster does not take into account some variable which are however present in the dataset such as; the price, title, designation and winery. As the taster will not be aware of the origin of the wine, features such as Country, Province and regions will not be explored for training the model.

This section presents the methodology for cleaning the preparing the dataset in order to maintain only the relevant features. Other features which are identified to be useful from the available variables are prepared and included in the final data to be explored and used in the training of the model.

2.1. Cleaning Repeated Featues

In this section we remove all features which are considered not to helpful in the training of the model for prediction of the points feature. This is based on the assumed scenario described above.

In [4]:

```
wines = wines.drop(columns = ['country','designation','price', 'province','region_1','region_2','taster_name', 'taster_twitter_handle','winery'])
```

2.2. Cleaning Duplicates

As it is unlikely for the different wines to have similar description and titles, in this section all duplicated rows for the description and title features are removed. However this is not done for the points variable as based on the description there are actually several wines with the similar points, and this is acceptable.

In [5]:

```
wines = wines.drop_duplicates(['description','title'])
```

2.3. Cleaning Missing Values

As the information for the variables to be used in the setup of the model are nominal, it is challenging to engineer possible values for missing data. For the purpose of this exercise, all rows with missing values for our relevant variables are removed. This is, in fact, only applicable for the variety column which has only one value missing as shown in the data description.

In [6]:

```
wines = wines.dropna(subset = ['description','points','title','variety'])
```

2.4. Getting features from the Title Variable

In the wines title there is a combination of different kinds of information. First of all, we can see that there is some information like designation or winery that is informartion that we already have in our dataset. By the way, there is an important feature that is not found anywhere else, the year since the wine has been aged.

To extract this information we use regular expressions to extract four numeric digits in the title. Proceading this way we can face the following situations:

- The year is the only four digit number in the title.

This situation is the ideal one and we dont have to worry about it.

- The year is in the title but there are also other four digit numbers.

Here we need to distinguish wich is the real year from other four digit numbers. In almost every example we have seen, the other four digit number corresponds to the winery foundation, that has to be before the wine was produced by the winery. This means that we can avoid this problem taking the highest number in the title, capped by 2017, that is the year in which tha data from this dataset was scrapped. We are aware that this method can lead to some errors, but it's very unlikely because some number between the real year of the wine and 2017 should be in the title.

- The year isn't in the title but there is some other four digit number.

This presents another problem because the only thing we can say about a four digit number is if it's likely to be the year of the wine, so this is what we do. Because of the wines are aged for 20 years at most, we only take years between 1997 and 2017. The 20 years rule has a few exceptions, but we prefear to not include those exceptions in our dataset than having some wrong years.

- There are any four digit numbers in the title.

This situation leads to having missing values in some wine years.

So, taking into acount what we have exposed we get the years with the following piece of code:

In [7]:

```
import re

def find_maxyear_inrange(text, first_year, last_year):
    four_digit_numbers = [int(number)
                          for number in re.findall('(\d{4})', text)    # Looks for four digit numbe
s
                          if first_year <= int(number) <= last_year]   # Check if four digits numbe
is in given range

    return max(four_digit_numbers, default = np.nan)    # Return the highest number satisfying the
previous conditions

wines['year'] = [find_maxyear_inrange(title, 1997, 2017)
                 for title in wines.title.values]
```

2.5. Getting Features from the Description Variable

To get features from the description our aim is to obtain some keywords that might appear in it. To achieve this goal, we remove words in the description that are too frequent in english, which means that they are not specific to or representative about the description of the wines. We also clean the stopwords from the description, which we are also not interested in. The stopwords have been obtained from the NLTK Corpus directory of most common stop words.

In [8]:

```
from nltk.corpus import stopwords
en_words = pd.read_csv('~/Desktop/Master/Wine_reviews/en50000words.txt', sep = ' ' , nrows = 1000)

# Helper function for pre-processing of descriptions
def preprocess_text(text):
    remove_words = en_words.word.values.tolist() + stopwords.words('english')
    return ' '.join( [word for word in text.split()
                      if word.lower()
                      not in remove_words] )

#get new description after preprocessing
wines['clean_description'] = [preprocess_text(description) for description in wines.description.val
ues]
```

Another feature generated from the descriptions is the length of each description. The correlation of this feature on the final points is checked during the data analysis stage. The generation of this feature is shown below.

In [9]:

```
wines = wines.assign(len_desc = wines['description'].apply(len))
```

After extracting the most important features and clearing the data, that's how our data looks like:

In [10]:

```
wines.head()
```

Out[10]:

	description	points	title	variety	year	clean_description	len_desc
0	Aromas include tropical fruit, broom, brimston...	87	Nicosia 2013 Vulkà Bianco (Etna)	White Blend	2013.0	Aromas include tropical fruit, broom, brimston...	172
1	This is ripe and fruity, a wine that is smooth...	87	Quinta dos Avidagos 2011 Avidagos Red (Douro)	Portuguese Red	2011.0	ripe fruity, wine smooth structured. Firm tann...	227
2	Tart and snappy, the flavors of lime flesh and...	87	Rainstorm 2013 Pinot Gris (Willamette Valley)	Pinot Gris	2013.0	Tart snappy, flavors lime flesh rind dominate....	186
3	Pineapple rind, lemon pith and orange blossom ...	87	St. Julian 2013 Reserve Late Harvest Riesling ...	Riesling	2013.0	Pineapple rind, lemon pith orange blossom arom...	199
4	Much like the regular bottling from 2012, this...	87	Sweet Cheeks 2012 Vintner's Reserve Wild Child...	Pinot Noir	2012.0	regular bottling 2012, rough tannic, rustic, e...	249

3.0. Exploratory Data Analysis

The main prediction label -points- will be analysed against relevant features such as the description, variety as well as the year which will be extracted from the title of each wine. The final input features to be incorporated in the training of the model as part of the description will be determined based on findings of the analysis. Below is a general statistical description of the remaining features after the preprocessing.

In [11]:

```
wines.describe(include='all')
```

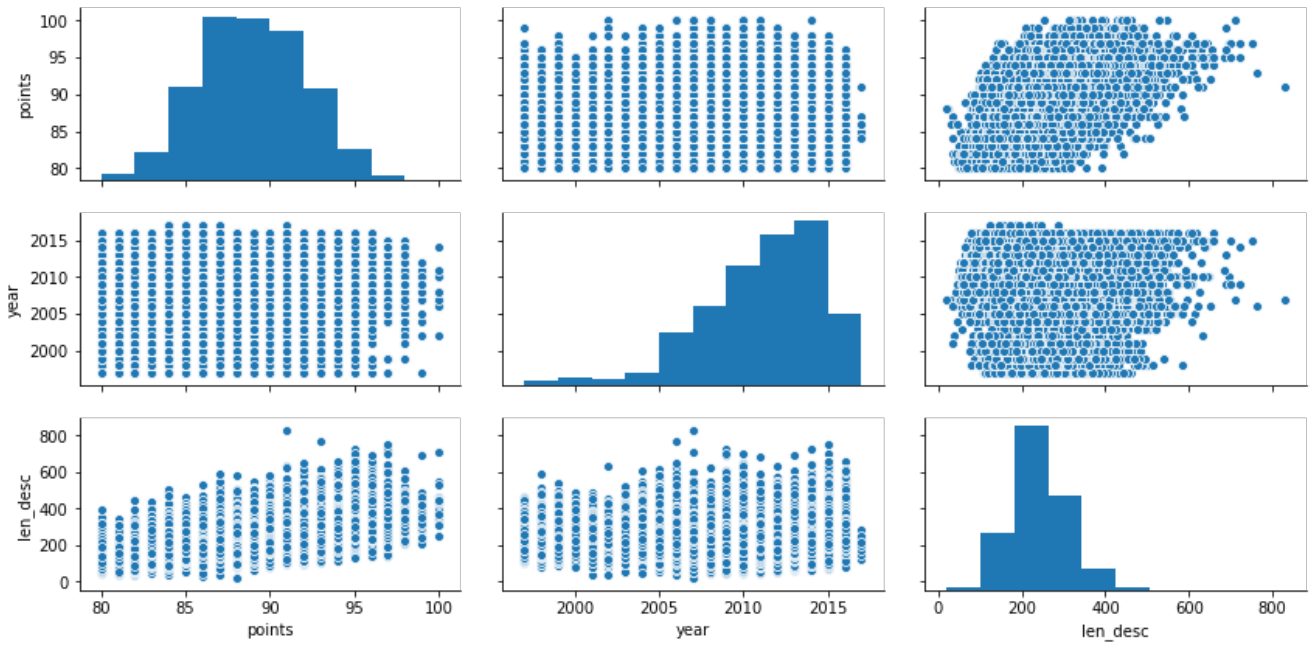
Out[11]:

	description	points	title	variety	year	clean_description	len_desc
count	119987	119987.000000	119987	119987	115478.000000	119987	119987.000000
unique	119954	NaN	118839	707	NaN	119950	NaN
top	Subtle aromas of acacia flower and a whiff of ...	NaN	Gloria Ferrer NV Sonoma Brut Sparkling (Sonoma...	Pinot Noir	NaN	Seductively tart lemon pith, cranberry pomegra...	NaN
freq	2	NaN	9	12278	NaN	2	NaN
mean	NaN	88.442240	NaN	NaN	2010.627323	NaN	242.809888
std	NaN	3.092928	NaN	NaN	3.537741	NaN	67.142001
min	NaN	80.000000	NaN	NaN	1997.000000	NaN	20.000000
25%	NaN	86.000000	NaN	NaN	2009.000000	NaN	197.000000
50%	NaN	88.000000	NaN	NaN	2011.000000	NaN	237.000000
75%	NaN	91.000000	NaN	NaN	2013.000000	NaN	283.000000
max	NaN	100.000000	NaN	NaN	2017.000000	NaN	829.000000

With the pairplot function from seaborn, a several plots are generated to give a preliminary general overview of the relationship between the features and to visualise the possible distributions. These are then developed further in the subsequent sections.

In [12]:

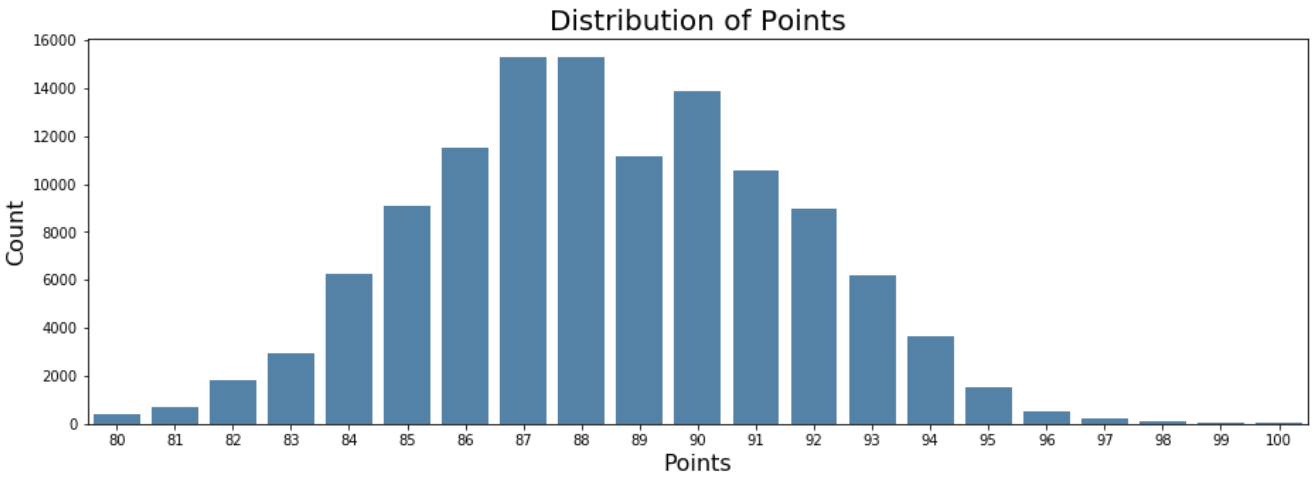
```
sn.pairplot(wines, palette='hls', height = 2,aspect = 2)
plt.show()
```



3.1. Points

In [13]:

```
plt.figure(figsize=(34,5))
plt.subplot(1,2,1)
plot = sn.countplot(x='points',data=wines, color="steelblue")
plot.set_title("Distribution of Points", fontsize= 20)
plot.set_xlabel("Points", fontsize = 16)
plot.set_ylabel("Count", fontsize =16)
plt.show()
```



The distribution of the points for the wines highlights the fact all the wines are graded between 80 and 100 points. The points seem almost distributed normally with most of the wine having points concentrated around the center.

The correlation between the points and the remaining features in our dataset is shown and visualised below:

In [14]:

```
wines.corr()
```

Out[14]:

	points	year	len_desc
points	1.000000	0.060926	0.562243
year	0.060926	1.000000	-0.072823
len_desc	0.562243	-0.072823	1.000000

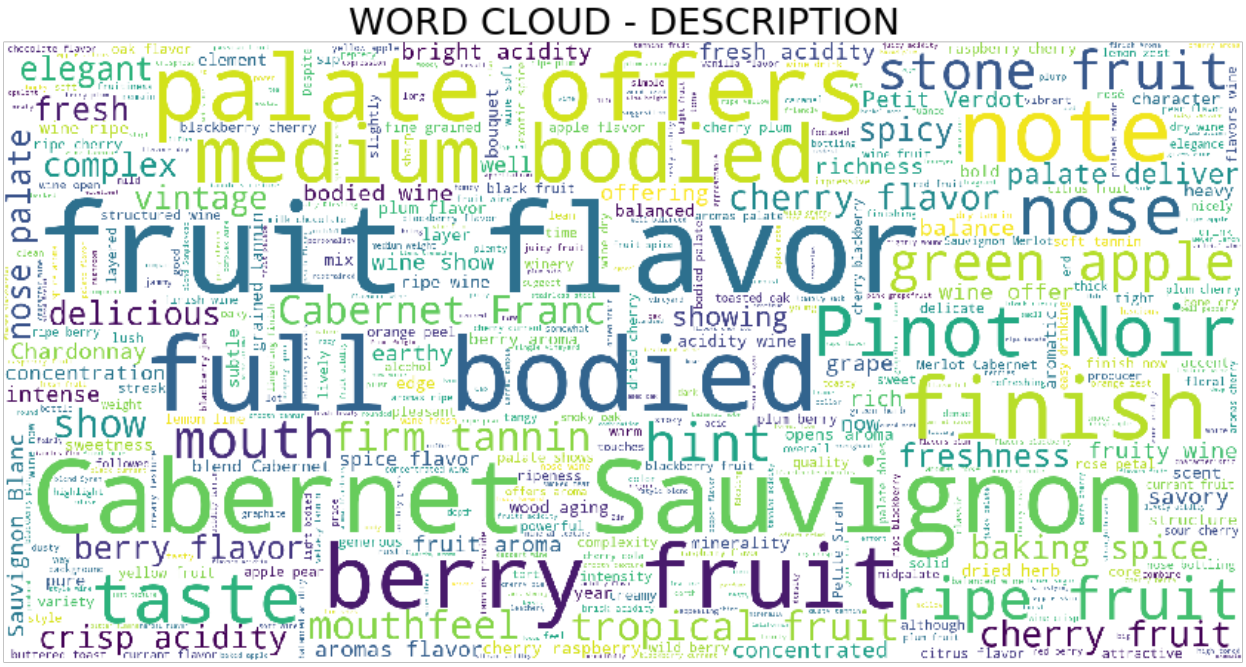
The major correlation observed here is between the length of the description and the points of the wine. This could be explained as the tasters tend to write more about a wine that they have a positive feeling about after tasting. This feature will be captured automatically in the model as part of the description. No significant correlation is observed between the year of the wine and the points allocated to it.

3.2. Description

A word cloud presentation of the words used in the descriptions is shown below to highlight the varying signifance of the words. Stopwords and very common engligh words have been removed to ensure the most specific words to the wine from he descriptions are considered for the training. This is evident in the wordcloud presentation below.

In [15]:

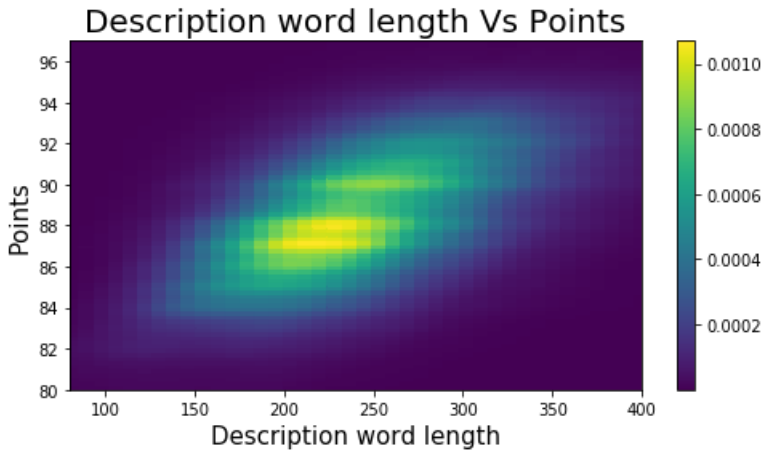
```
wordcloud = WordCloud(background_color ='white', max_words = 500,
max_font_size = 200,width =2000,height=1000,random_state=50,).generate(" ".join(wines['clean_descri
ption'].astype(str)))
wordcloud
fig = plt.figure(figsize=(15,15))
plt.imshow(wordcloud)

plt.title("WORD CLOUD - DESCRIPTION", fontsize=25)
plt.axis('off')
plt.show()
```



As shown in the correlation plots there exists some correlation between the length of the descriptions and the points. The plot below highlights this correlation.

In [16]:

```
from scipy.stats import gaussian_kde

def plot_density_map(x, y, nbins = 100):
    k = gaussian_kde([x,y])
    xi, yi = np.mgrid[x.min():x.max():nbins*1j, y.min():y.max():nbins*1j]
    zi = k(np.vstack([xi.flatten(), yi.flatten()]))
    plt.pcolormesh(xi, yi, zi.reshape(xi.shape))
    plt.colorbar()

plt.figure(figsize=(8,4))
plot_density_map(wines.len_desc, wines.points)
plt.xlim(80,400)
plt.ylim(80,97)
plt.title('Description word length Vs Points', fontsize = 20)
plt.xlabel('Description word length', fontsize = 15)
plt.ylabel('Points', fontsize = 15)
plt.show()
```

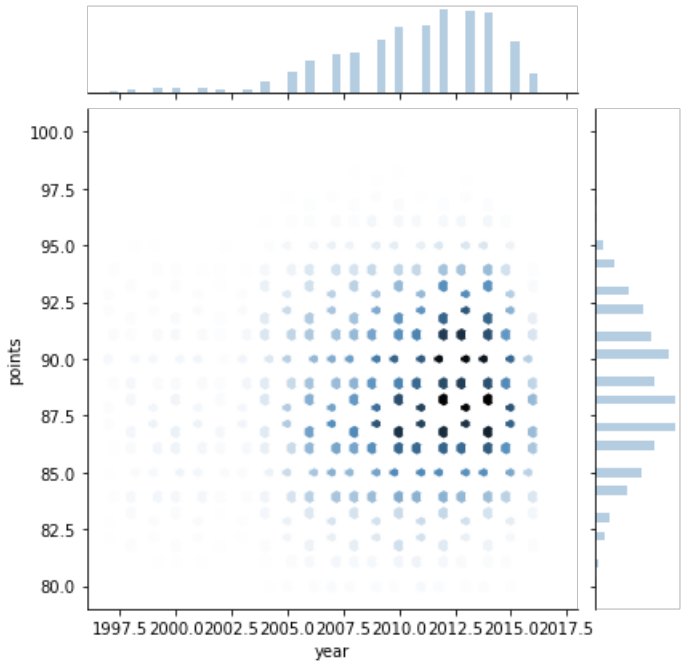


The above plot presents a minor quantity of the wines which can be classified as outliers. This is an insignification proportion with respect to the entire dataset and as such they will not be processed specially.

3.3. Year

In [17]:

```
sn.jointplot(x=('year'), y=('points'), data=wines, kind="hex", color='steelblue')
plt.show()
```



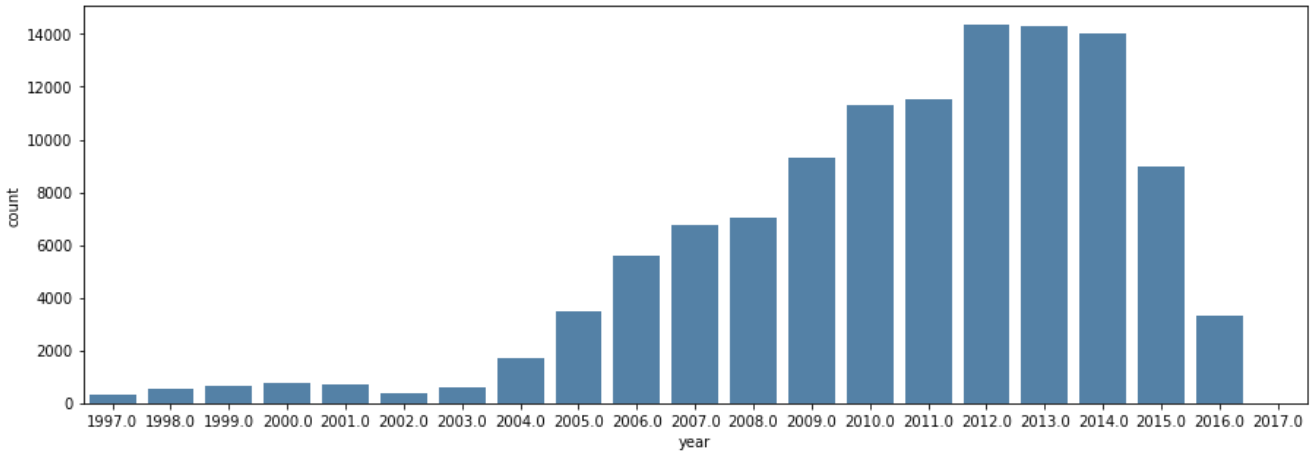
From the visualization above, it is noticed that a large proportion of the wines considered in the wine review dataset are from of the last decade. This is further highlighted by the distrubtion chart below.

In [18]:

```
plt.figure(figsize=(15,5))
sn.countplot(x='year', data=wines, color="steelblue")
plot.set_title("Distribution of Year", fontsize= 20)
plot.set_xlabel("year", fontsize = 16)
plot.set_ylabel("Count", fontsize =16)
plt.show
```

Out[18]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```



4.0 Model Set-Up

The job of a sommelier is to provide a meaningful description of the wine and then, according to the description, give some punctuation of the taste and aromas that the wine can offer. We want to develop a model that is able to do this second step; predicting the points awarded by a sommelier according to the description given. With the results of this model, we expect to be able to say if the punctuation provided by the sommelier is fair according to the descriprion he also provided.

Our model will be a neural network which particularities will be described later. Our model will need to be trained, but it also needs to be tested in order to see how good our model is.

4.1 Splitting Data into Train and Test Data

So, first of all, we need to divide our dataset in train data and test data.

10% of the final cleaned data is reserved to be used for evaluating the model after the training.

In [19]:

```
# helper function for splitting data
def train_test_split(data, train_frac = 0.8):
    shuffled_data = wines.sample(frac=1)
    split = int(train_frac * len(shuffled_data.index))
    train = shuffled_data[:split]
    test = shuffled_data[split:]
    return (train, test)

#split dataset in train and test
wines_train, wines_test = train_test_split(wines, train_frac = 0.9)
```

4.2 Pre-Processing Descriptions

After spliting the data, we need to prepare the data in order that the neural network can interpret it with ease. First of all, we are going to deal with the description. We use the class Tokenizer provided by the library keras wich allows us to preprocess text. A total number of 3000 words is considered for the training. This is a hyperparameter which has been decided on finally and has been observed to increase the the models output. This value also gives a good balance between fairly good results and the a good amount of parameters for the neural network.

With the method fit_on_texts, we obtain the most common words in our train descriptions and then, we use the method texts_to_matrix to one-hot encode the descriptions weighted by the "tf-idf" formula.

In [20]:

```
import tensorflow as tf
import keras

#Get 3000 most common words in the descriptions
description_tokenizer = keras.preprocessing.text.Tokenizer(num_words = 3000)
description_tokenizer.fit_on_texts(wines_train.clean_description)

#One-hot encode the descriptions with the words previously obtained, using the tf-idf mode
description_train = description_tokenizer.texts_to_matrix(wines_train.clean_description, mode =
'tfidf')
description_test = description_tokenizer.texts_to_matrix(wines_test.clean_description, mode =
'tfidf')

#Set points as the labels which our predictions must fit
points_train = wines_train.points
points_test = wines_test.points
```

```
Using TensorFlow backend.
```

4.3 Pre-Processing of Varieties

Before a sommelier tastes a wine, he/she gets information about the variety and the vintage of it, so this is information that we also want to include in the model. Both features are categorical, thus we will use one-hot encoding to prepare the data for our model. So, following a similar procedure that we did with the description, we first find the most common ocurrences in those features and, then, we one-hot encode our data according to those most common varieties and years.

In [21]:

```
import collections

def fit_mostcommon(data, max_items = None):
    most_common = collections.Counter(data.dropna()).most_common(max_items)
    return [item[0] for item in most_common]

def onehot_encoding(data, keys):
    return np.array([[int(item == key) for key in keys] for item in data])

# Onehot encoding of the top 50 varieties
variety_keys = fit_mostcommon(wines_train.variety, max_items = 50)

variety_train = onehot_encoding(wines_train.variety, variety_keys)
variety_test = onehot_encoding(wines_test.variety, variety_keys)

# Onehot encoding of all years previously selected
year_keys = fit_mostcommon(wines_train.year)

year_train = onehot_encoding(wines_train.year, year_keys)
year_test = onehot_encoding(wines_test.year, year_keys)
```

4.4. Neural Network Implementation

Now that we have our data ready, we are going to build our neural network. We will create three separate input layers for each feature, and each one will have a fully connected hidden layer to process the data from each individual feature. Then, we will merge

those hidden layers concatenating them, and use this new merged layer as input for a new fully connected layer. Finally, we add one last layer without activation function and with only one neuron that provides the output.

As we are dealing with a regression problem, we found that the best activation function for the hidden layers was the sigmoid, but the results were very similar with ReLU. To compute the gradient we tried diferent loss functions and some of them provided similar results, one of the best performing ones was mean squared error, which we chose as our loss function not because it is the best performing but because we have a better intuition of what it does.

In [22]:

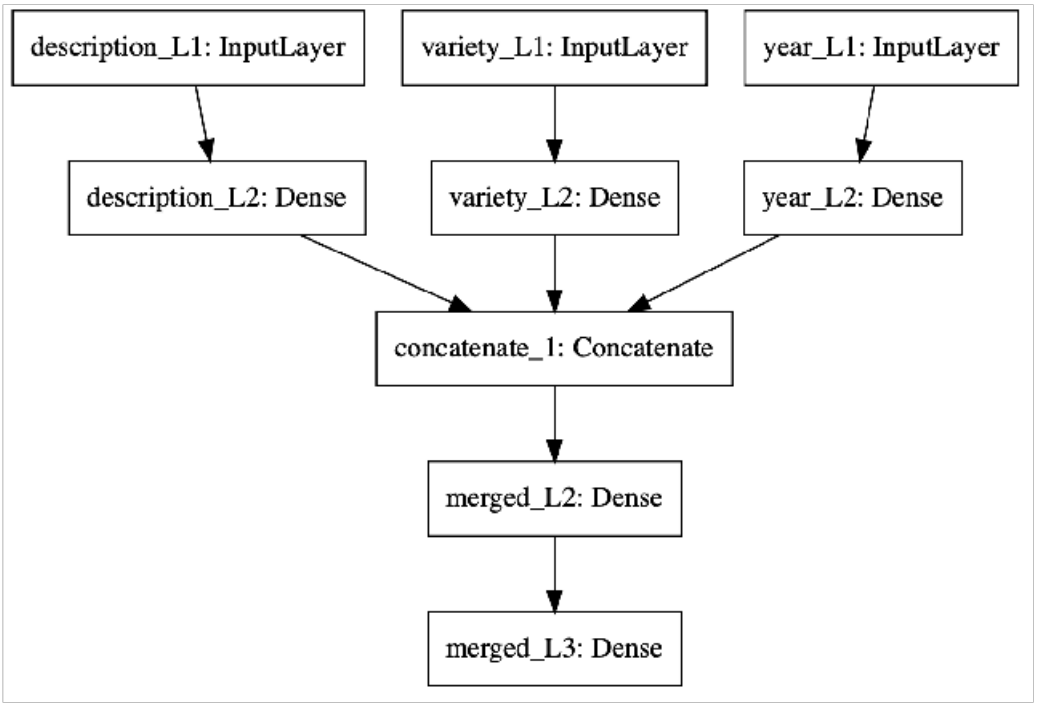
```
description_L1 = keras.layers.Input(shape = (description_train.shape[1],), name = 'description_L1')
description_L2 = keras.layers.Dense(128, activation = 'sigmoid', name = 'description_L2')(descripti
on_L1)

variety_L1 = keras.layers.Input(shape = (variety_train.shape[1],), name = 'variety_L1')
variety_L2 = keras.layers.Dense(16, activation = 'sigmoid', name = 'variety_L2')(variety_L1)

year_L1 = keras.layers.Input(shape = (year_train.shape[1],), name = 'year_L1')
year_L2 = keras.layers.Dense(8, activation = 'sigmoid', name = 'year_L2')(year_L1)

merged_L1 = keras.layers.Concatenate()([description_L2, variety_L2, year_L2])
merged_L2 = keras.layers.Dense(64, activation = 'sigmoid', name = 'merged_L2')(merged_L1)
merged_L3 = keras.layers.Dense(1, name = 'merged_L3')(merged_L2)

neural_network = keras.Model(inputs = [description_L1, variety_L1, year_L1],
                             outputs = [merged_L3])

neural_network.compile(loss = 'mse',
                       optimizer = keras.optimizers.RMSprop(),
                       metrics = ['mae'])

neural_network.summary()
```

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
description_L1 (InputLayer)     (None, 3000)         0
__________________________________________________________________________________________________
variety_L1 (InputLayer)         (None, 50)           0
__________________________________________________________________________________________________
year_L1 (InputLayer)            (None, 21)           0
__________________________________________________________________________________________________
description_L2 (Dense)          (None, 128)          384128      description_L1[0][0]
__________________________________________________________________________________________________
variety_L2 (Dense)              (None, 16)           816         variety_L1[0][0]
__________________________________________________________________________________________________
year_L2 (Dense)                 (None, 8)            176         year_L1[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 152)          0           description_L2[0][0]
                                                                 variety_L2[0][0]
                                                                 year_L2[0][0]
__________________________________________________________________________________________________
merged_L2 (Dense)               (None, 64)           9792        concatenate_1[0][0]
__________________________________________________________________________________________________
merged_L3 (Dense)               (None, 1)            65          merged_L2[0][0]
==================================================================================================
Total params: 394,977
Trainable params: 394,977
Non-trainable params: 0
__________________________________________________________________________________________________
```

To give a better picture of what our model does, we can visualize the structure of the neural network in the following diagram.

In [23]:

```
import matplotlib.image as mpimg
diagram = mpimg.imread('nn_diagram.png')
plt.figure(figsize=(12,8))
plt.imshow(diagram)
plt.axis('off')
plt.show()
```



4.5. Training Neural Network

Now that the model set up is completed, the model is trained with the data. We define a callback function that prints the epoch every 10 epochs in order to track the progress. We train the model over a certain number of epochs and save all the progress in a history that we will use latter to show the training progress.

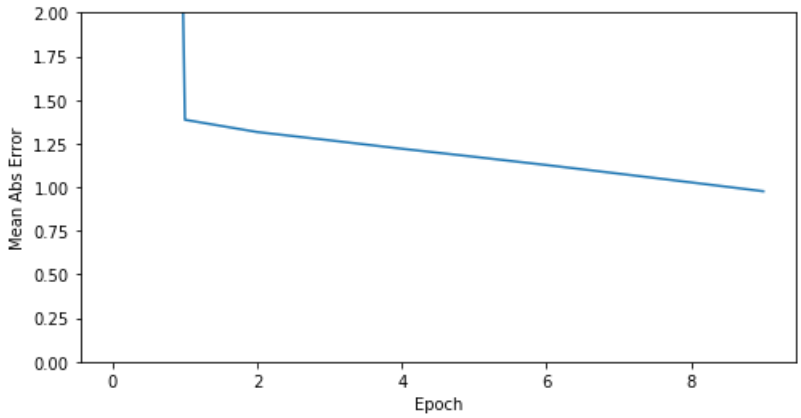
In [24]:

```
class PrintEpoch(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs):
        print('Epoch {0} / {1}'.format(epoch, EPOCHS))

EPOCHS = 10

history = neural_network.fit([description_train, variety_train, year_train], points_train, epochs =
EPOCHS,
                             validation_split = 0.2, verbose = 0, callbacks = [PrintEpoch()])
```

```
Epoch 0 / 10
Epoch 1 / 10
Epoch 2 / 10
Epoch 3 / 10
Epoch 4 / 10
Epoch 5 / 10
Epoch 6 / 10
Epoch 7 / 10
Epoch 8 / 10
Epoch 9 / 10
```

In [25]:

```
def plot_history(history):
    plt.figure(figsize=(8,4))
    plt.xlabel('Epoch')
    plt.ylabel('Mean Abs Error')
    plt.plot(history.epoch, np.array(history.history['mean_absolute_error']))
    plt.ylim([0, 2])

plot_history(history)
```



4.6. Validation of the results

Now that we have our model trained, we need to check if it predicts the points well. First of all we predict the points with our model and compare it with the real values.

In [26]:

```
points_predictions = neural_network.predict([description_test, variety_test, year_test]).flatten()
points_predictions = np.round(points_predictions)

pd.DataFrame({'points predictions': points_predictions, 'real points': points_test}).head(10)
```

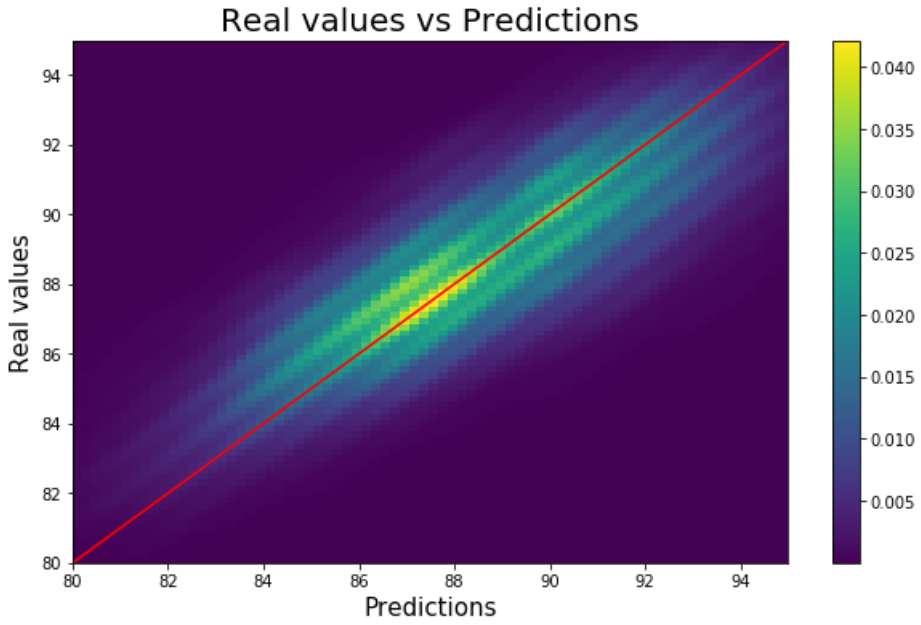
Out[26]:

	points predictions	real points
87835	88.0	88
72705	89.0	87
104629	88.0	86
106402	90.0	91
122081	93.0	92
42879	88.0	89
104125	84.0	84
113959	89.0	90
20582	87.0	86
63501	86.0	85

To evaluate better this results we plot our predictions versus their actual values. If the model was perfect, which means that will always be predicting the correct results, all the points should be over the blue line. This way, we can see how wrong is our model in every predictions. We used colors to represent the density of points in each region of the plot to show that the majority of the predictions are close to their desired values.

In [27]:

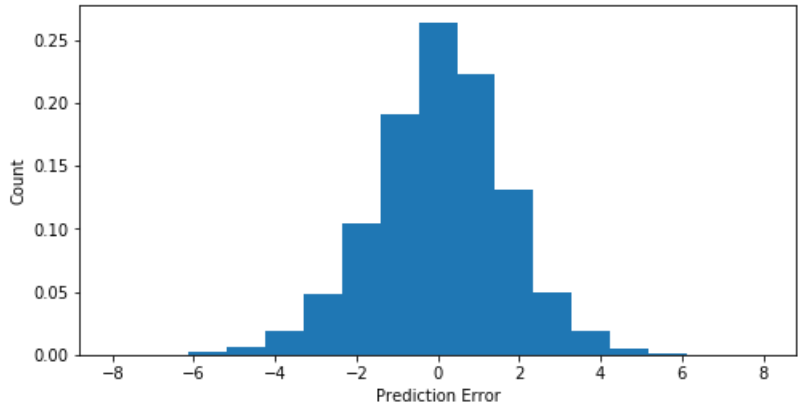
```
plt.figure(figsize=(10,6))
plot_density_map(points_test, points_predictions)
plt.xlim(80,95)
plt.ylim(80,95)
plt.title('Real values vs Predictions', fontsize = 20)
plt.xlabel('Predictions', fontsize = 15)
plt.ylabel('Real values', fontsize = 15)
plt.plot([0, 100], [0, 100], c = 'red')
plt.show()
```



Finally, we plot the density distribution of the errors in the predictions in order to give an extra measure of how close to the real results our model is making predictions.

In [28]:

```
def plot_error_hist(test_labels, predictions):
    error = points_predictions - points_test
    plt.figure(figsize=(8,4))
    plt.hist(error, bins = 17, density = True)
    plt.xlabel("Prediction Error")
    plt.ylabel("Count")
    plt.show()
    print('Mean absolute error = {0}'.format(abs(error).mean()))

plot_error_hist(points_test, points_predictions)
```




```
Mean absolute error = 1.3214434385299683
```

5.0. Conclusions

We made a model to predict the points awarded to wines by sommeliers according to a description they also provided. We consider that our model is making good predictions according to the evaluation we did of the results, as we got a mean absolute error around 1.3 points which is close enough to the actual value. If we compare this error with the one we would get giving the mean points as the result of our model, which is the simplest model we can think about, we can get an extra measure of how well is our model predicting points.

In [29]:

```
abs(wines.points - wines.points.mean()).mean()
```

Out[29]:

```
2.535302210724676
```

As we pointed out earlier, this model could be useful to detect irregularities in the points awarded by the sommeliers. This can be used as back up to verify the final points given to a wine by s Somemelier based on their provided descriptions. This could be done by checking the cases in which the sommelier is giving a result very different that the one which our model predicts and then use other criteria to be able to say if a sommelier is not being fair doing his job.

Of course the model can still be improved, this is just a simple model that deals with data that usually is hard to process, like texts in the case of the description, or cathegorical values in the case of the varieties and years. In this sense, we have learned a lot while doing this project, as we went from knowing almost anything about machine learning to be able to build our first neural network and be able to make predictions with it.