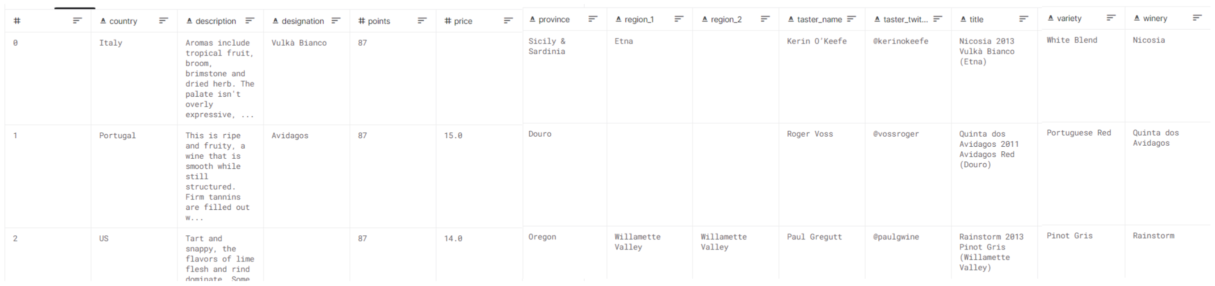
Machine Learning Project
Task 2 Report

Albert Garaev,
Ksenia Novikova,
Mukhammadsodik Khabibulloev

04.01.2022

1 Data Pre-processing

We download the dataset wich contains wine reviews and other information about wine as a pandas DataFrame. The Figure1 shows the view of our data.



#	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twit...	title	variety	winery
0	Italy	Aromas include tropical fruit, broom, brimstone and dried herb. The palate isn't overly expressive, ...	Vulkà Bianco	87		Sicily & Sardinia	Etna		Kerin O'Keefe	@kerinokeefe	Nicosia 2013 Vulkà Bianco (Etna)	White Blend	Nicosia
1	Portugal	This is ripe and fruity, a wine that is smooth while still structured. Firm tannins are filled out w...	Avidagos	87	15.0	Douro			Roger Voss	@vossroger	Quinta dos Avidagos 2011 Avidagos Red (Douro)	Portuguese Red	Quinta dos Avidagos
2	US	Tart and snappy, the flavors of lime flesh and rind dominate. Some		87	14.0	Oregon	Willamette Valley	Willamette Valley	Paul Gregutt	@paulgwine	Rainstorm 2013 Pinot Gris (Willamette Valley)	Pinot Gris	Rainstorm

Figure 1: View of the dataset

We need to extract the wine's years from the title column and add them as a new column to our data. We write a function below.

```python
Year = []
for value in wines['title']:
        result = re.search(r'19\d{2}|20\d{2}', value)
        if result:
                Year.append(float(result.group()))
        else: Year.append(np.nan)

wines['year'] = Year
```

We can plot a histogram for each columns. We group the numerecal into bins as usual, for ither types of data we count each unique value and order them on the x-axis by descending count. The Figure 2 shows histogram for numerical data.

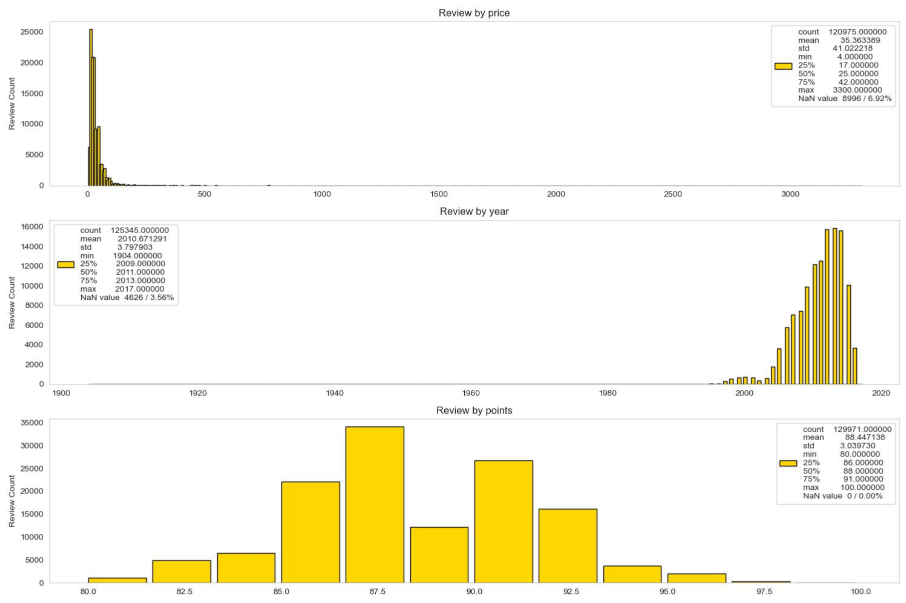
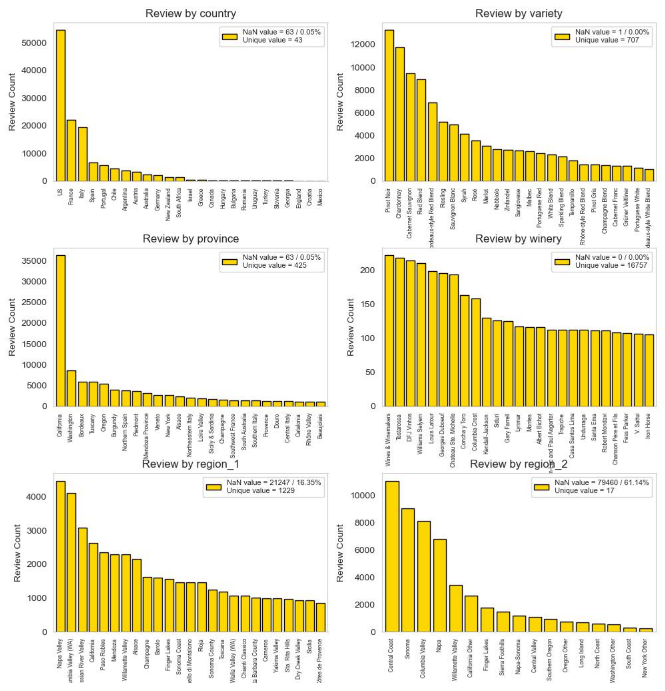
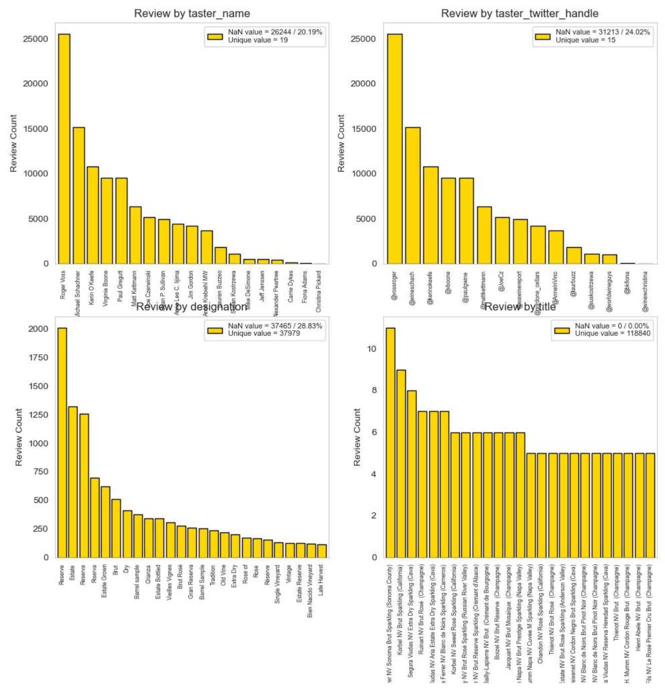


Figure 2: Histograms for numerical data

The Figure3 shows histogram for other data.



(a)



(b)

Figure 3: Histograms for other data

We use *pandas.isna*(*obj*), which detect missing values for an array-like object, to show how many of the values are missing for each column. *pandas.DataFrame.describe*() generates descriptive statistics for numeric data such as count, mean, std, min, max as well as lower (default is 25), 50 and upper (default 75) percentiles, the 50 percentile is the same as the median [1]

We need to transform our data into real numbers. Also we need to handle with missing(NaN) values.

We implement *nullscan*(*df*), where we perofrm the nullscan for a dataframe. The *pandas.isna*(*obj*) returns boolean value. Dataframe (the size of the original dataframe) contain value True in null-cells and value False in the other cells. After we interpret True as 1 and False as 0. We can see after using *nullscan*(*df*) that the dataset contains missing values and we need to handle them.

```
def nullsearch(wines_check):
        wines_nulls = wines_check.isna()
        nulls_per_col = wines_nulls.sum(axis=0)
        nulls_per_col /= len(wines_check.index)

        with plt.style.context('dark_background'):
                        figure_5, (ax1, ax2) = plt.subplots(nrows=2, ncols=1,
                        sharex=True, figsize=(6, 8))
                vir = matplotlib.cm.get_cmap('viridis')
                colormap = matplotlib.colors.ListedColormap([vir(0), 'white'])
                sns.heatmap(wines_check.isnull(), cmap=colormap, cbar=False,
                yticklabels=False, ax=ax1)

                nulls_per_col.plot(kind='bar', color='white', x=nulls_per_col.values,
                y=nulls_per_col.index, ax=ax2, width=1, linewidth=1,
                edgecolor='black', align='edge', label='NaN value rate')

                ax2.set_ylim((0, 1))
                # centered labels
                labels = wines_check.columns
                ticks = np.arange(0.5, len(labels))
                ax2.xaxis.set(ticks=ticks, ticklabels=labels)

                ax2.spines['top'].set_color('black')
                ax2.spines['right'].set_color('black')

                #very small amounts of NaN values per column
                na_ticks = ticks[(nulls_per_col > 0) & (nulls_per_col < 0.05)]
                if (len(na_ticks) > 0):
                ax2.plot(na_ticks, [0, ] * len(na_ticks), 's', c='white', markersize=10,
                label='Very few missing values')

                figure_5.suptitle('NaN values evaluation', fontsize=30, y=1.05)
                ax2.legend()
                figure_5.tight_layout()
                plt.show()

nullsearch(wines)
```

The Figure 4 shows the rate of null-values in each column.

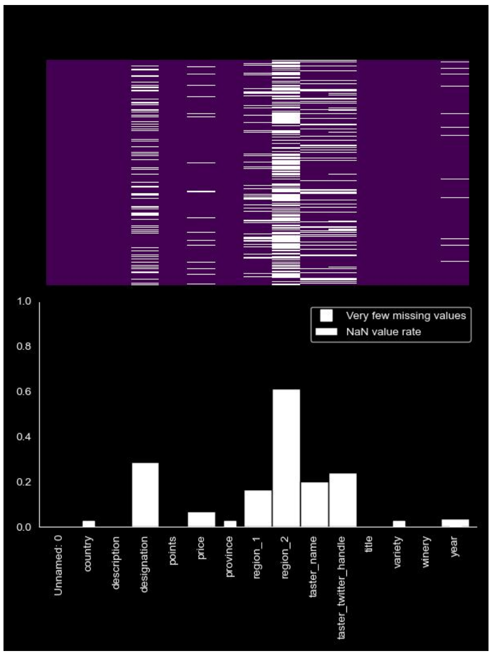


Figure 4: Amount of NaN values in each column

There are many methods to handle Null Values. We choose method of droping columns, wich contains a specified amount of null-values, if the number of missing values is very high and the column isn't too relevant.
The Figure 5 shows the rate of null-values after dropping some columns.

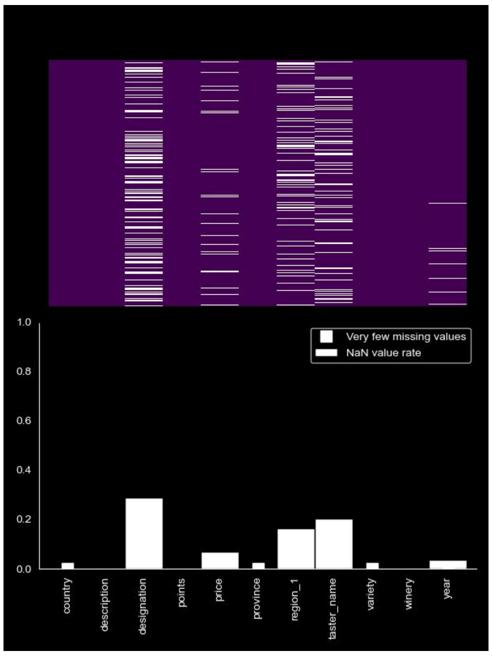


Figure 5: Amount of NaN values in each column after columns dropping

After that we change NaN-values to the "Unknown". The Figure 6 shows the columns after changing.

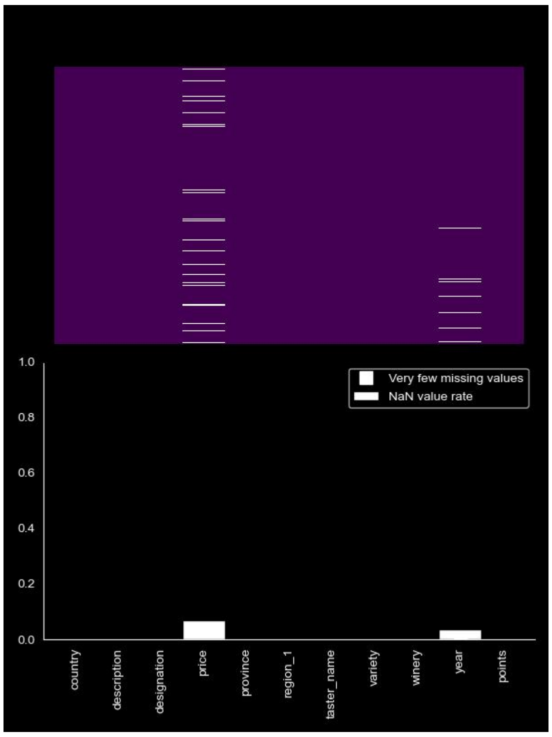
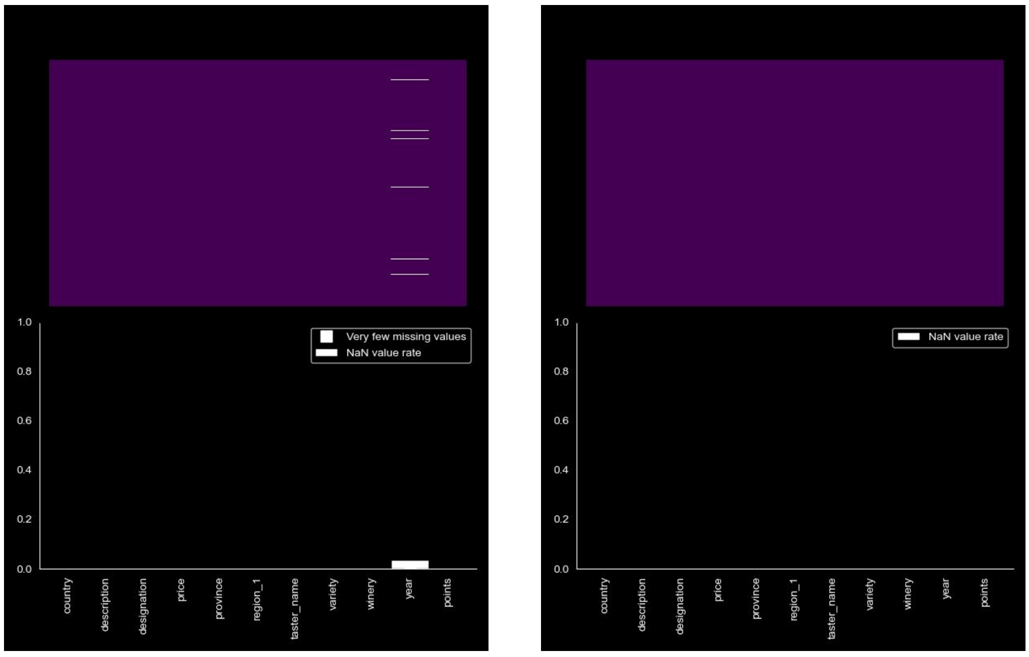


Figure 6: Columns after changing to the "Unknown"-values

At last, we predict null-values at the columns "price" and "year" using KNN method. The Figure 7 shows the results for columns "Price" and "Year".



(a)

(b)

Figure 7: Predicted values for columns a)"Price", b)"Year"

After we can transofm the categorical data into numbers. We use *LabelEncoder* for data tranformation and create dictionary with our transformed into numbers categorical data. This method is easy to impliment. After we transform text columns using Word2Vec algorithm. This algorithm was used due to ease of use, open source code and speed. We download a pre-trained word2vec model by google $GoogleNews - vectors - negative300.bin$. After that we can transform the dataset row by row, checking is data categorical or text and using functions wich was described above.

Pre-processing our data. Type of wine is a text data type. In this case, the Word2Vec function is used for processing. At first, the vector is initialized to zero. After that, words are searched for in the model dictionary. If the search is successful, the vector becomes numeric, otherwise (the word was not found in the dictionary) the vector remains zero. The best solution might be to increase the dictionary so that no exceptions are thrown and all text variables (in our case the type of wine) are converted to numeric vectors.

Outliers have a huge impact on data quality. We should definitely detect outliers for several reasons. Outliers distort the overall picture and thus reduce accuracy. Probably the best known way to detect outliers is using a z-score. We use *scipy.stats.zscore*() for this task.

2 Regression

We implement a class *RidgeRegression*.

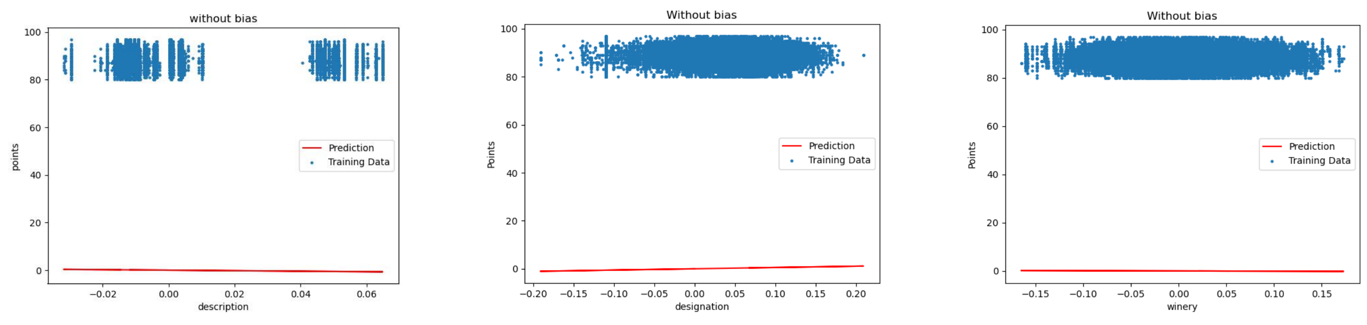
```
class RidgeRegression:
        def init(self, C):
                self.C = C
                self.w = None

        def fit(self, X, y):
                self.w = np.linalg.inv(X.T @ X + (1 / self.C) * np.eye(X.shape[1])) @ (X.T @ y)

        def predict(self, X):
                return np.dot(X, self.w)

        def calc_metrics(self, X_train, y_train, X_test, y_test):
                self.fit(X_train, y_train)
                train_error = self.calc_error(X_train, y_train)
                validation_error = self.calc_error(X_test, y_test)
                return train_error, validation_error
```

We use Principal component analysis (PCA). This is one of the main ways to reduce the dimension of data, losing the least amount of information. The method reduces the linear dimension by using singular value decomposition to project it into a lower-dimensional space. In this case, for each function before SVD is applied, the input is centered but not scaled.
Thus we can see in Figure8 scatter plot that shows the projected value of the column and line wich is result of of ridge regression.



(a) (b) (c)

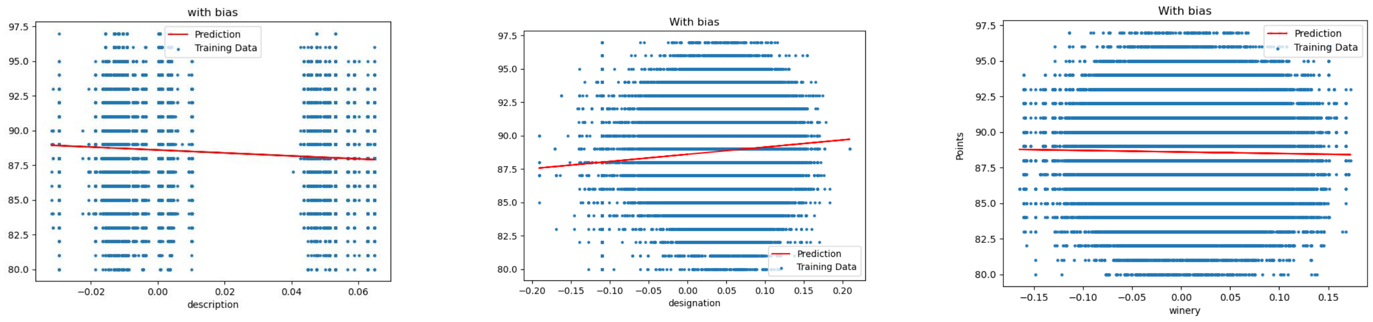
Figure 8: Scatter plot for ridge regression without bias

We implement a new class *RidgeRegressionBias* that uses the unchanged class *RidgeRegression*:

```
class RidgeRegressionBias(RidgeRegression):
        def fit(self, X, y):
                X = np.hstack((np.ones((X.shape[0], 1)), X))
                self.w = np.linalg.inv(X.T @ X + (1 / self.C) * np.eye(X.shape[1])) @ (X.T @ y)

        def calc_error(self, X, y):
                X = np.c_[np.ones(X.shape[0]), X]
                predictions = self.predict(X)
                mse = mean_squared_error(y, predictions)
                return mse
```

Bias is the ability of the algorithm model to adjust to the target dependence.
Thus we can see in Figure 9 scatter plot that shows the projected value of the column and line wich is result of ridge regression with bias.



(a) (b) (c)

Figure 9: Scatter plot for ridge regression with bias

We can now estimate the performance of our regressor using a fivefold cross validation and root mean square error (MSE) as a measure of performance. We implement *mean_squared_error*() wich calculates mean squared error regression loss. We can see results in Table 1

Type	K1	K2	K3	K4	K5
Train errors in each Fold	5.53	5.49	5.63	5.49	5.60
Validation errors in each Fold	5.64	5.79	5.23	5.79	5.36

Table 1: Values of mean square error (MSE)

As we know each component w_j of the weight vector is forced to be small in magnitude by the regularization term in ridge regression. The corresponding input feature x_j can never contribute much to the final prediction, if the varies only slightly compared to the other features, although this may be important. In contrast, the corresponding feature's effect on the final prediction will be disproportionately large, if it takes values in a rather big interval. We can standardize features by removing the mean and scaling to unit variance to solve this problem. We use class sklearn *StandardScaler*() to implement standardization.

After that we compute mean squared error (MSE) again. We can see new results in Table 2

Type	K1	K2	K3	K4	K5
Train errors in each Fold	5.42	5.38	5.52	5.39	5.48
Validation errors in each Fold	5.53	5.66	5.14	5.64	5.31

Table 2: Values of mean square error (MSE) using StandartScaler

As wee can see value of MSE is smaller than before. So we can assume that in our case standardization helps us to recieve better results.

The essence of the Forward-Stepwise Selection is as follows: We start with a model that does not contain variables (called a null model). Then we start adding the most significant variables one by one. Until the specified stopping rule is reached or until all the variables in question are included in the model. In our case, the stopping rule will be the inclusion of 3 variables in the model, since we found that this is enough to achieve an adequate estimate of the model.

We have a simple dataset which consists of scalar inputs and regression targets. We can see the plot of the dataset in Figure 10. We can assume looking at the plot that this data was generated using standard normal distribution or t-distribution.

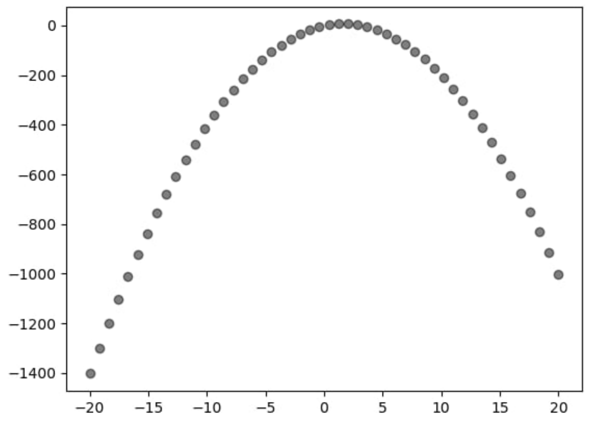


Figure 10: Plot of the dataset

We apply our implementation of ridge regression on this data. The Figure 11 the resulting line in the same plot as the data.

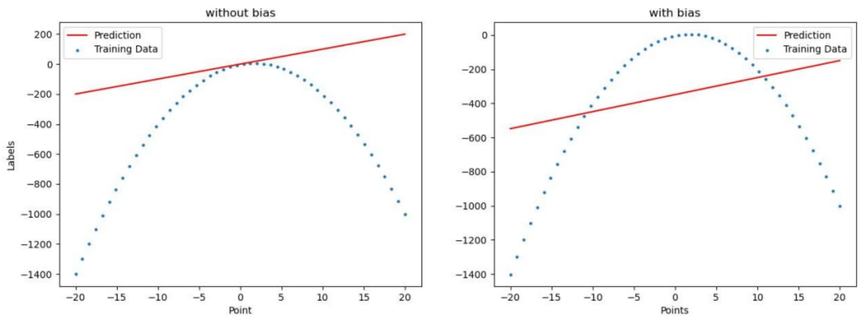


Figure 11: Plot of the dataset with resulting line

Ridge regression does not really work well in the case when the relation between input and target variables is not linear for a dataset. In this case the value of MSE is large (train error = 152084.58644527048 and test error = 103133.98900313386) and model can not be called adequate.

We need to find suitable ϕ in formula: $f(x) = w^T\phi(x) + b = \sum_{j=1}^{d} w_j\phi(x)_j + b$

We can see at the plot in Figure 10 that our data looks like parabola. We know the equation for this function: $y = ax^2 + bx + c$, we also know the values of points. In this case, we constracted the system of equations and find approximate values $a \approx -3,\ b \approx 10\ and\ c \approx -2$. After this expression ϕ becomes $\phi(x) = -3x^2 + 10x - 2$.

Thus $f(x) = w^T(-3x^2 + 10x - 2) + b = \sum_{j=1}^{d} w_j(-3x_j^2 + 10x_j - 2) + b$

Now the MSE close to zero: train error = 2.576008724130151e-07 and test error = 1.225935120189919e-07. The Figure 12 shows the plot with pridictions line.

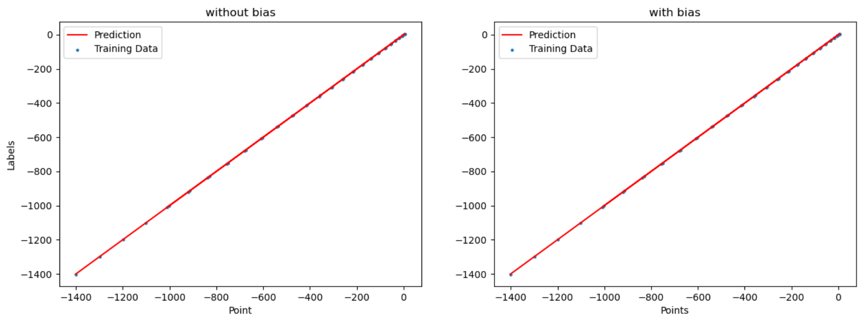


Figure 12: Plot of the dataset with resulting line

References

[1] Wes McKinney. *pandas: powerful Python data analysis toolkit, Version 1.3.5, Dec 12, 2021.*