Artificial Intelligence - MSc

CS6501 - MACHINE LEARNING APPLICATIONS

Instructor: Enrique Naredo

CS6501_Etivity-3

In [ ]:
```
#@title Current Date
Today = '2021-10-22' #@param {type:"date"}
```

In [ ]:
```
#@markdown ---
#@markdown ### Enter your details here:
Student_ID = "17246067" #@param {type:"string"}
Student_full_name = "James Larkin" #@param {type:"string"}
Student_ID = "17238889" #@param {type:"string"}
Student_full_name = "Karl Mullane" #@param {type:"string"}
Student_ID = "16170571" #@param {type:"string"}
Student_full_name = "Gerard Holihan" #@param {type:"string"}
Student_ID = "17236444" #@param {type:"string"}
Student_full_name = "Sean Mortimer" #@param {type:"string"}
Student_ID = "17230004" #@param {type:"string"}
Student_full_name = "Eoin Halpin" #@param {type:"string"}
#@markdown ---
```

In [ ]:
```
#@title Notebook information
Notebook_type = 'Etivity' #@param ["Example", "Lab", "Practice", "Etivity", "Assignment", "Exam"]
Version = Final #@param ["Draft", "Final"] {type:"raw"}
Submission = True #@param {type:"boolean"}
```

1. Introduction

A regression problem is a type of supervised learning problem. The goal of a regression model, much like that of a classification problem is to create a model "that can predict the value of the dependent attribute from the attribute variables" (Shukla, 2021). However, despite being similar to classification, regression is different. The goal of classification is to predict a class label, whereas the goal of a regression is to predict a continuous number. The simplest way to distinguish between them is to examine whether or not there is some element of continuity in the result. If there is continuity present, it is a regression problem (Muller and Guido, 2016).

The goal of this etivity will be to build one synthetic dataset, and use one 'real-world' dataset (California_housing) in an attempt to address regression problems and to handle outliers. The methods used to handle outliers in this etivity will be discussed later.

Outliers pop up regardless of how careful one is when collecting their data and can be sources of great frustration for data scientists. An outlier can be defined as "a data point that is noticeably different from the rest" (Wardukar, 2020). Essentially, outliers signify that there are errors in measurement, there may have been variable that weren't considerd when collecting the data or maybe the data was collected poorly (Warudkar, 2020). Unfortunately, there is no single best method for detecting outliers. Each method has it's own benefits and drawbacks. The methods we are using in this etivity are the Z-Score Based Technique, the Interquartile Range Based Filtering method (IQR), the Percentile based technique and the Winsorization method.

Outlier Techniques for Detection and Removal

The Z-score based technique is a useful tool in the detection of outliers. The Z-score will help us to understand whether a value is bigger or smaller than the mean and also how far away it is from the mean. In a normal distribution it is assumed that 95% of all the data values will lie within either plus or minus two standard deviations $(\pm 2\sigma)$ away from the mean. Therefore, if the Z-score of a particular data value is greater than 2, this would lead us to assume the data value is very different from the others and it can subsequently be classified as an outlier. (Maini, 2020).

The IQR Based Filtering method is concerned with finding the difference between the first and third quartiles of a dataset. One way of thinking about the IQR is to imagine it as being a spread of the bulk of the data. In thinking of the IQR in this way, it implies that outliers are observations that are far from the main concentration of the data. In the IQR Based Filtering method "outliers are commonly defined as any value $1.5(IQR)$ less than the first quartile or $1.5(IQR)$ greater than the third quartile (Albon, 2018).

Winsorization is a method in which one can remove the influence of outliers in a dataset. This can be done in one of two ways, either by assigning a lower weight to the outlier or by changing the value so that it is close to other values in the set. Within the Winsorization method, the data

values are modified (statisticshowto.com, 2016). In Winsorizing, "any value of a variable above or below a percentile k on each side of the variables' distribution is replaced with the value of the k-th percentile itself" (Horsch, 2021). The Winsorization method is less extreme than other methods of dealing with outliers because it recodes outliers rather than just cutting them altogether.

Methods

Linear Regression is a relatively simple supervised machine learning algorithm. It is a common and useful method for making preditions about the target vector, especially when it is a quantitative value. It "assumes that the relationship between the features and the target vector is approximately linear" (Albon, 2018). In essence, linear regression can be used to predict the value of a variable based on the value of another variable (ibm.com, n.d.). One of the major advantages of using linear regression is that it is a very interpretable model.

Logistic Regression is another supervised machine learning technique used for classification. It allows the coder to make predictions about the probability that an observation is of a certain class. Despite its name, it is an algorithm used for classification, not regression and it is important not to get it confused with linear regression (Albon, 2018).

XGBoost is a package in Python and applies grdient boosting to a arge scale problem. It is a tree-based model and is widely used for supervised learning (Muller and Guido, 2016). The XGBoost Regressor that we are using in this etivity can be used directly for regression predictive modelling.

The K-Nearest Neighbour (KNN) algorithm is an algorithm that "uses the k nearest observations (according to some dsitance metric) to predict the missing value" (Albon, 2018). It can be used for both classification and regression problems. As a classifier, the KNN is considered somewhat lazy. This is due to the fact that it doesn't make predictions by training the model, observations are predicted to be the class of that of the largest proportion of the k nearest observations. For regression, the target is predicted by local interpolation of the targets associated of the nearest neighbours in the training set.

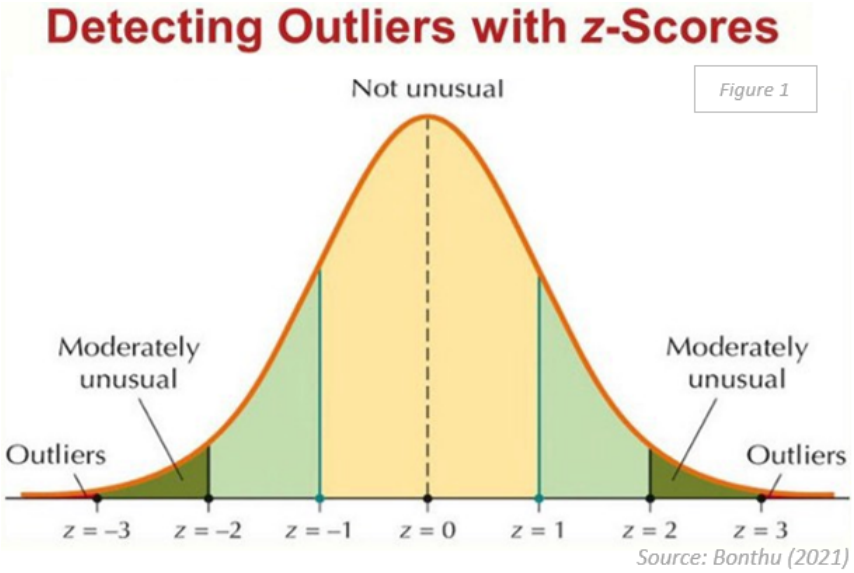
1.1 Background

1.2 Methodology

Before getting started on running statistical models and analysis me must first organize the datasets i.e., data cleaning. This is called the preprocessing stage. NaN stands for 'Not a Number' (represented by None in Python) and this is a special floating point-value used for representing missing data. Detecting and removing these missing data points is important so that our code can work properly e.g., the sklearn methods. There are two common ways to deal with NaN values (Păpăluță, 2020). The first method is to drop the whole row respective to where the NaN value is. Although a viable method, it proposes the risk in losing valuable information in the other variables within this tuple. Also, if many NaN values existed, removing the data rows could drastically reduce the number of observations which would take away from the accuracy of the models' outputs. The second method involves filling the NaN value with another value. One way in which we can do this is by filling it with 0. However, this is only a good option in some datasets e.g., it is viable for money spent on something but bad for age. Another way is metrics imputation whereby the NaN is replaced by a metric such as the mean or the median etc. KNN can also be used to fill the NaN with the value calculated by mean of the closest k samples in the dataset to the NaN value.

The second part of the preprocessing stage is to deal with outliers. These are datapoints that vary dramatically from the others i.e., values that fall out of the normal distribution (Hoppen, 2021). These outliers can cause bias in the results if the purpose of the analysis is to interpret the entire sample as a whole. The four techniques to deal with outliers that will be used in this analysis are the Z-score based technique, IQR based filtering, percentile-based technique and winsorization.

Z Score Based Technique

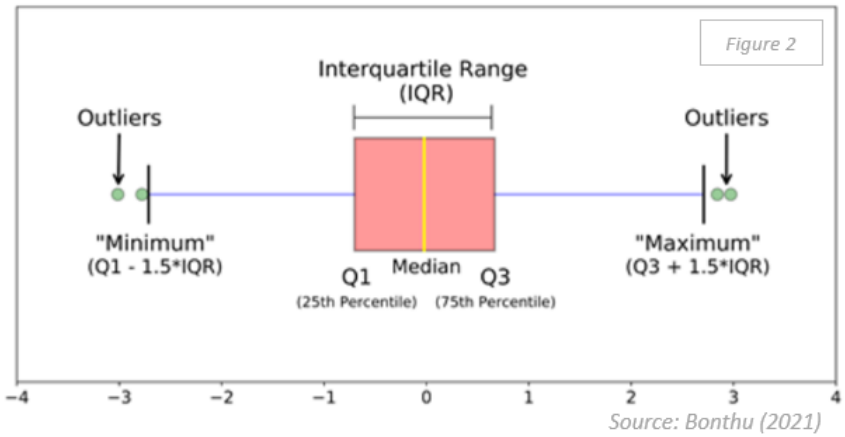


This example employs a normal distribution whereby outliers are defined as any data points whose Z-score meet either of the following conditions: $Z < -3$ or $Z > 3$. To do this, the Z-score formula is applied to all data points. The formula for a Z-score is as follows:

$Z = \frac{x-\mu}{\sigma}$

where x is the observation, μ is the mean and σ is the standard deviation. In this example, the threshold to classify outliers is 3. However, the threshold can be modified within our python code.

IQR Based Filtering



Source: Bonthu (2021)

The interquartile range (IQR) is a measure of statistical dispersion displayed by a boxplot. It refers to the middle 50% calculated by the 75th percentile minus the 25th percentile i.e., $Q3 - Q1$. The process starts with sorting the dataset in ascending order so that the IQR can be calculated. From here, a lower bound and upper bound is calculated and any lies below or above respectively can be classified as an outlier.

Formula:

$$LB = Q1 - 1.5xIQR$$

$$UB = Q3 + 1.5xIQR$$

Once again, in this example the threshold is influenced by the value 1.5 which can be modified within our python code.

Percentile-based technique

This technique is very similar to the one used above as it uses percentiles. However, this technique locates the upper and lower bounds directly. This analysis locates the values that fall below 10% of the data and lies above 90%. These are then classified as outliers and removed or 'trimmed' from the original dataset.

Winsorization

Introduced by Tukey & McLaughlin in 1963, winsorizing is a technique to deal with outliers that is often used in research papers (Horsch, 2021). Like the previous two techniques explained, this also includes percentiles in the process. Datapoints that lie above or below the k-th percentile are replaced with the k-th percentile value itself. This is generally preferred over the percentile-based technique as it maintains the datapoints/information rather than 'trimming' them altogether. For the purpose of this analysis k will be set to the 5th percentile.

The four regression models that will be used in this analysis are the linear, logistic, XGBoost and KNN regression models. We have covered the linear, logistic and XGBoost classification models in previous etivities. However, in this analysis these models will be used for regression purposes instead of classification. This means that our models can now predict the relationship between our dependent variable and independent variables as well as forecasting the dependent variable.

Logistic Model

$$log(\frac{\pi}{1-\pi}) = \beta_0 + B_1x_1 + B_2x_2 + \ldots B_mx_m$$

XGBoost Model

$$f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$$

$$f_m(x) = F_{m-1}(x) + h_m(x)$$

K-nearest neighbours

K-nearest Neighbours (KNN) is a supervised machine learning algorithm. It assumes that the data exits in clusters or within close proximity (Medium.com, 2021). The value for K is set at the number of data points that the new prediciton will be related to. The similarity between these K data points and the new points is calculated using the Euclidean distance which follows the following formula:

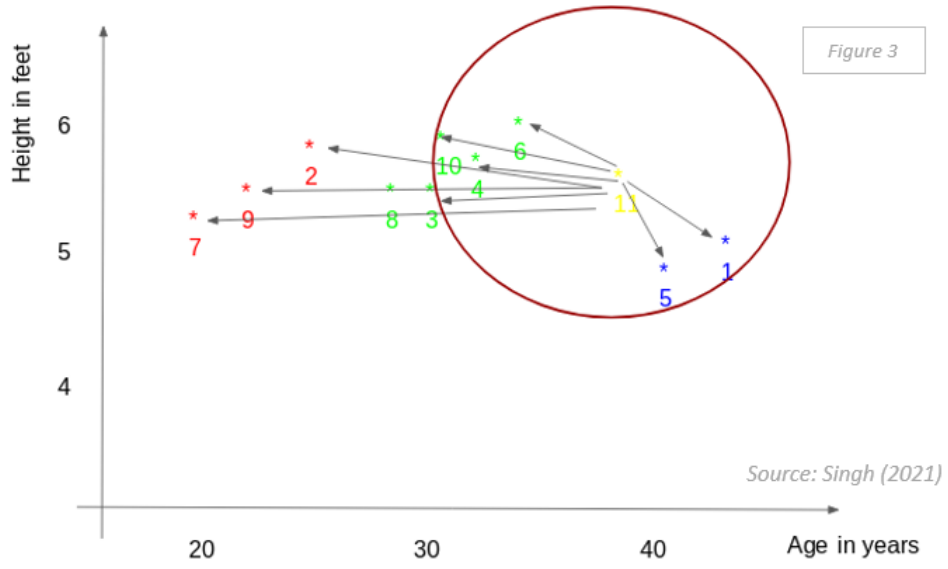
$$d(x,y) = d(y,x) = \sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2 + \ldots (y_n - x_n)^2}$$

$$= \sum_{n=1}^{\infty} (y_i - x_i)^2$$

note: different distance metrics can also be used such as manhattan distance.

This denotes the distance from point x to point y. The output is the average of the K values. The K-nearest data points in the training data are selected by comparing to the Euclidean distance. The optimal value for K is chosen by trying different values and seeing which has the least error.

K Nearest Neighbours



In the example in the image above, K is set to 5, and so uses the mean of the five closest points to estimate the new point.

Linear Regression

This is based off a linear relationship between two variables x and y such that

$$y = \beta_0 + \beta_1 x + \epsilon$$

β_0 is the intercept which is the average value of y when x is 0. β_1 is the slope which is the expected change in y with a 1-unit increase in x

Imports

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tru
e).
```

In [ ]:
```
#!pip uninstall scikit-learn -y
!pip install -U scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (1.0.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn) (1.0.1)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn) (1.4.1)
Requirement already satisfied: numpy>=1.14.6 in /usr/local/lib/python3.7/dist-packages (from scikit-learn) (1.19.5)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn) (3.0.0)
```

In [ ]:
```
from collections import Counter
from pandas.plotting import scatter_matrix
from scipy import *
from scipy import stats
from scipy.stats.mstats import winsorize
from six.moves import urllib
from sklearn import linear_model
from sklearn.datasets import fetch_california_housing
from sklearn.datasets import make_regression
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.linear_model import BayesianRidge
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import r2_score
from sklearn.metrics import recall_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from tabulate import tabulate
from xgboost import XGBRegressor
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import os
```

```
import pandas
import pandas as pd
import random
import requests
import seaborn as sns
import tarfile
import warnings
import xgboost as xgb
warnings.filterwarnings("ignore")
```

DATASETS

2. Synthetic Dataset

2.1 Data Preparation

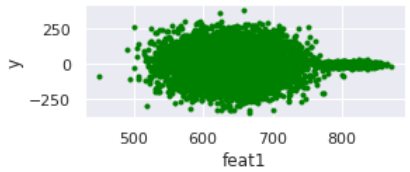
In [ ]:
```
#to replicate experiments
np.random.seed(7)
n_samples = 10000
```

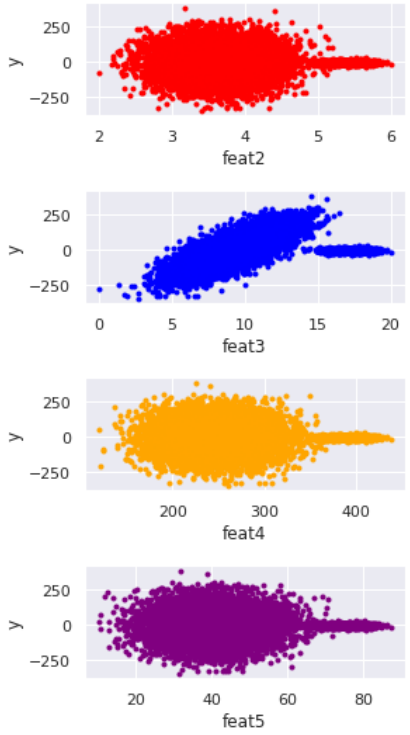
In [ ]:
```
#making a regression dataset with:
  # number of samples: 10000
  # number of features: 5
  # noise: 65

X, y = make_regression(n_samples=n_samples,
                       n_features=5,
                       n_informative=1,
                       noise=65,
                       random_state=0)
```

In [ ]:
```
feat1 = X[:,0]
feat2 = X[:,1]
feat3 = X[:,2]
feat4 = X[:,3]
feat5 = X[:,4]
```

2.2 Data Visualisation

In [ ]:
```
import seaborn as sns

sns.set(rc={'figure.figsize':(9,5)})

plt.subplot(3,2,1)
plt.scatter(feat1, y, color='green', marker='.')
plt.xlabel("feat1")
plt.ylabel("y")
plt.show()

plt.subplot(3,2,2)
plt.scatter(feat2, y, color='red', marker='.')
plt.xlabel("feat2")
plt.ylabel("y")
plt.show()

plt.subplot(3,2,3)
plt.scatter(feat3, y, color='blue', marker='.')
plt.xlabel("feat3")
plt.ylabel("y")
plt.show()
#1st col of x plotted against y

plt.subplot(3,2,4)
plt.scatter(feat4, y, color='orange', marker='.')
plt.xlabel("feat4")
plt.ylabel("y")
plt.show()

plt.subplot(3,2,5)
plt.scatter(feat5, y, color='purple', marker='.')
plt.xlabel("feat5")
plt.ylabel("y")
plt.show()
```





2.2.1 Adding the Outlier Data

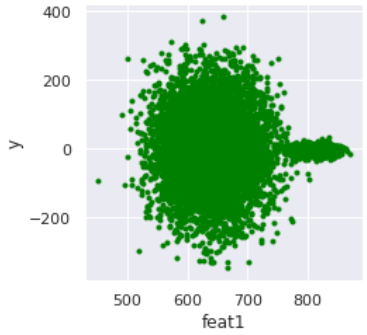
In [ ]:

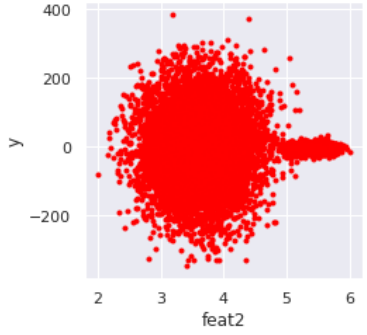
```
# add outlier data
n_outliers = int(np.round(n_samples*0.1))
th = 4
X[:n_outliers] = th + 0.5 * np.random.normal(size=(n_outliers, 1))
y[:n_outliers] = -th + 10 * np.random.normal(size=n_outliers)
```

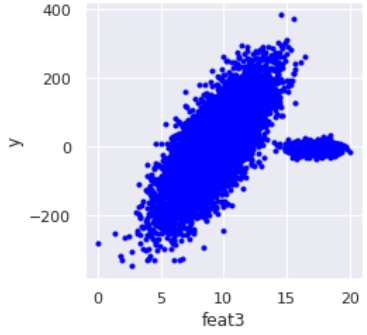
2.2.2 Plotting the Synthetic Regression Dataset - With Outliers

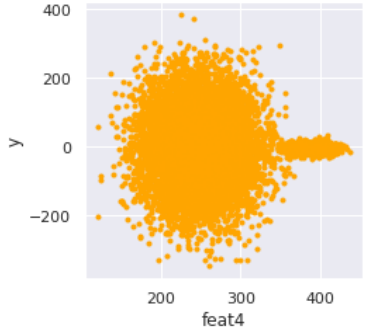
In [ ]:

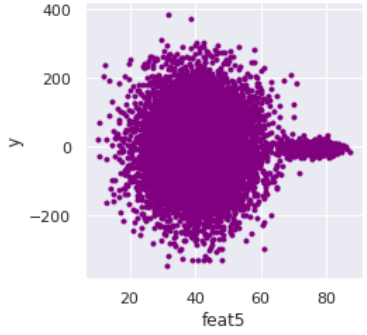
```
sns.set(rc={'figure.figsize':(12,8)})

plt.subplot(2,3,1)
plt.scatter(feat1, y, color='green', marker='.')
plt.xlabel("feat1")
plt.ylabel("y")
plt.show()

plt.subplot(2,3,2)
plt.scatter(feat2, y, color='red', marker='.')
plt.xlabel("feat2")
plt.ylabel("y")
plt.show()

plt.subplot(2,3,3)
plt.scatter(feat3, y, color='blue', marker='.')
plt.xlabel("feat3")
plt.ylabel("y")
plt.show()

plt.subplot(2,3,4)
plt.scatter(feat4, y, color='orange', marker='.')
plt.xlabel("feat4")
plt.ylabel("y")
plt.show()

plt.subplot(2,3,5)
plt.scatter(feat5, y, color='purple', marker='.')
plt.xlabel("feat5")
plt.ylabel("y")
plt.show()
```











2.2.3 Scaling

In [ ]:

```
# scale feature 1: 450-870 (float)
# scale feature 2: 2-6 (integer)
# scale feature 3: 0-20 (integer)
# scale feature 4: 120.56-436.92 (float)
# scale feature 5: 10.22-87.15 (float)
# scale output var: 150000-2000000 (integer)
```

In [ ]:

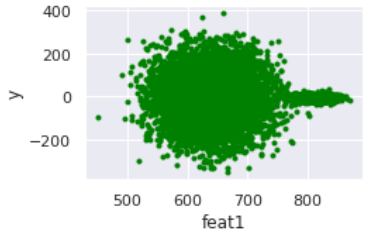
```
# your code here
# Scale Feature 1:
```

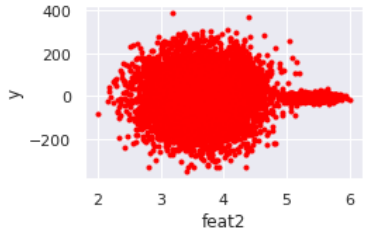
```
b_value1, t_value1= 450, 870
feat1 = np.interp(feat1, (feat1.min(), feat1.max()), (b_value1, t_value1))

#Scale Feature 2:
b_value2, t_value2= 2, 6
feat2 = np.interp(feat2, (feat2.min(), feat2.max()), (b_value2, t_value2))

#Scale Feature 3:
b_value3, t_value3= 0, 20
feat3 = np.interp(feat3, (feat3.min(), feat3.max()), (b_value3, t_value3))

#Scale Feature 4:
b_value4, t_value4= 120.56, 436.92
feat4 = np.interp(feat4, (feat4.min(), feat4.max()), (b_value4, t_value4))

b_value5, t_value5= 10.22, 87.15
feat5 = np.interp(feat5, (feat5.min(), feat5.max()), (b_value5, t_value5))
```

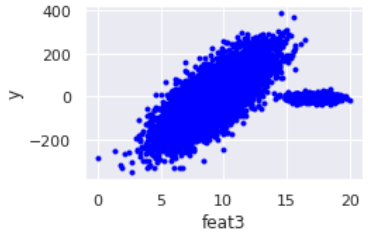
2.2.4 Plotting the Synthetic Regression Dataset - After Scaling

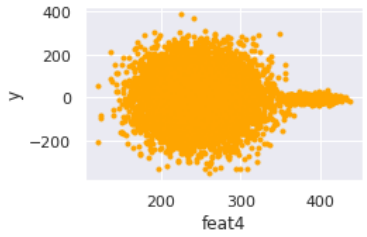
In [ ]:

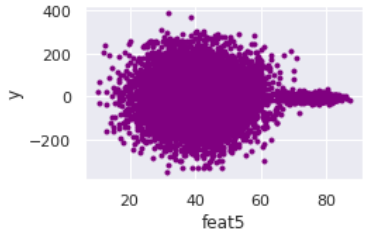
```
#plot
sns.set(rc={'figure.figsize':(12,5)})


plt.subplot(2,3,1)
plt.scatter(feat1, y, color='green', marker='.')
plt.xlabel("feat1")
plt.ylabel("y")
plt.show()

plt.subplot(2,3,2)
plt.scatter(feat2, y, color='red', marker='.')
plt.xlabel("feat2")
plt.ylabel("y")
plt.show()

plt.subplot(2,3,3)
plt.scatter(feat3, y, color='blue', marker='.')
plt.xlabel("feat3")
plt.ylabel("y")
plt.show()

plt.subplot(2,3,4)
plt.scatter(feat4, y, color='orange', marker='.')
plt.xlabel("feat4")
plt.ylabel("y")
plt.show()

plt.subplot(2,3,5)
plt.scatter(feat5, y, color='purple', marker='.')
plt.xlabel("feat5")
plt.ylabel("y")
plt.show()
```











2.2.5 Dataframe - Synthetic Regression Dataset

In [ ]:
```
df = pd.DataFrame({'feature1': feat1, 'feature2': feat2, 'feature3': feat3, 'feature4': feat4, 'feature5': feat5, 'y': y})
df.head()
```

Out[ ]:

	feature1	feature2	feature3	feature4	feature5	y
0	845.112931	5.744720	18.839955	416.477928	82.153664	4.848880
1	799.264025	5.274422	16.702830	378.817945	72.949023	-13.091182
2	809.868188	5.383195	17.197115	387.528134	75.077918	7.353174
3	817.834662	5.464911	17.568451	394.071742	76.677270	-3.131487
4	792.396975	5.203983	16.382741	373.177397	71.570392	3.516674

In [ ]:
```
df.describe()
```

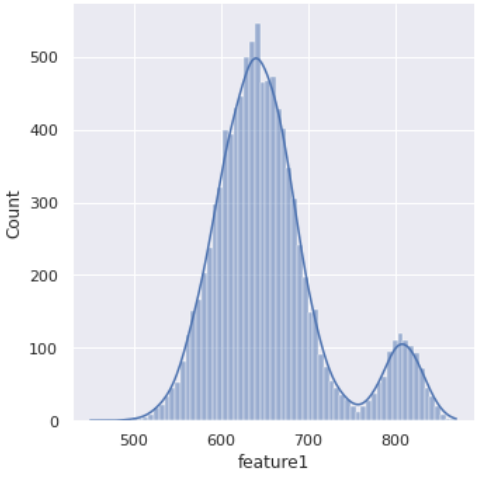
Out[ ]:

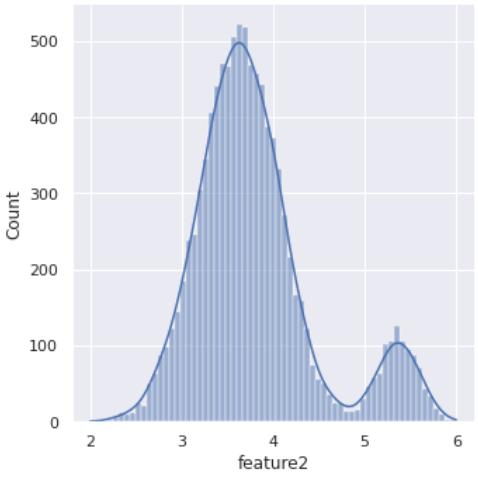
	feature1	feature2	feature3	feature4	feature5	y
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	655.742626	3.798837	10.022558	261.630866	44.195058	-0.709401
std	65.323021	0.669866	3.046308	53.487921	13.045625	93.810756
min	450.000000	2.000000	0.000000	120.560000	10.220000	-347.837176
25%	613.396456	3.374018	8.061117	227.085684	36.028459	-57.789358
50%	644.761369	3.686174	9.515242	252.788926	41.985883	-2.906362
75%	679.465172	4.041219	11.153091	281.902839	48.801024	57.966711
max	870.000000	6.000000	20.000000	436.920000	87.150000	386.024044

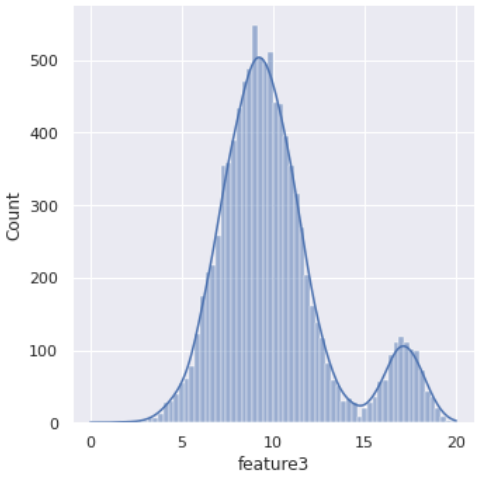
2.2.6 Distribution - Synthetic Regression Dataset

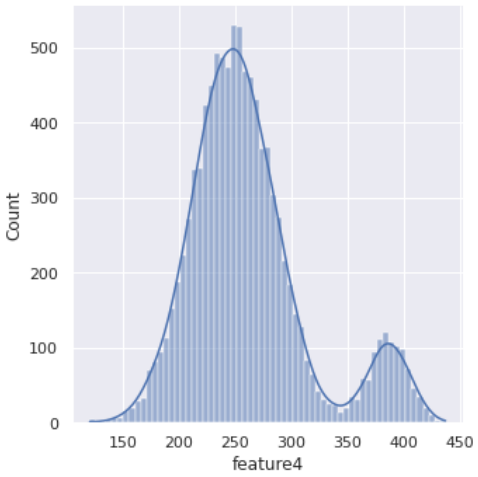
In [ ]:
```
f, axs = plt.subplots(1,2,
                      figsize=(9,5),
                      sharey=True)
plt.subplot(5, 2, 1)
sns.displot(df['feature1'], kde=True)
plt.subplot(5, 2, 2)
sns.displot(df['feature2'], kde=True)

sns.displot(df['feature3'], kde=True)

sns.displot(df['feature4'], kde=True)

sns.displot(df['feature5'], kde=True)

sns.displot(df['y'], kde=True)
```

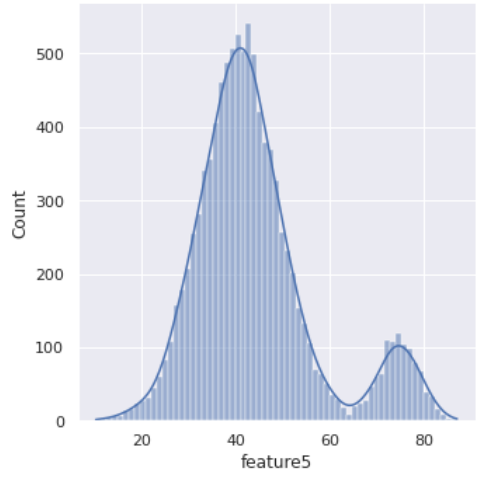
Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7f8f9da79850>
```

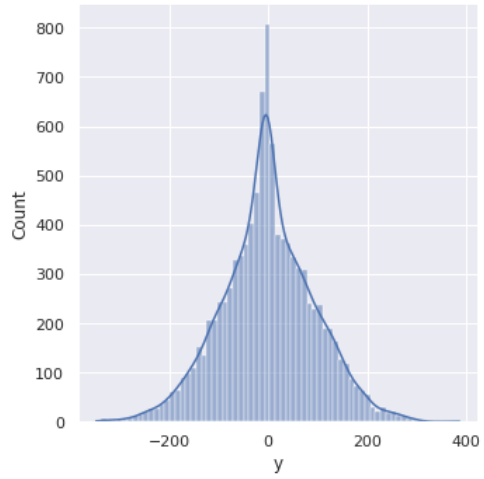

Count
0
100
200
300
400
500
500
600
700
800
feature1


Count
0
100
200
300
400
500
2
3
4
5
6
feature2


Count
0
100
200
300
400
500
0
5
10
15
20
feature3


Count
0
100
200
300
400
500
150
200
250
300
350
400
450
feature4





2.3 Detecting NANs

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
feature1    0
feature2    0
feature3    0
feature4    0
feature5    0
y           0
dtype: int64
```

2.4 Dealing with Imbalanced Dataset

This section is inapplicable as the synthetic dataset created was balanced and there were no NANs present or anomalies.

2.5 Outlier Techniques

2.5.1 Z-Score Based Technique

In [ ]:
```
df_z = df.copy()
```

In [ ]:
```
#finding the boundary values
threshold = 3
HA = df_z['y'].mean() + threshold*df_z['y'].std()
LA = df_z['y'].mean() - threshold*df_z['y'].std()
print("highest allowed",HA)
print("lowest allowed",LA)
```

```
highest allowed 280.7228678849592
lowest allowed -282.1416700947043
```

In [ ]:
```
#finding the outliers
df_z[(df_z['y'] > HA) | (df_z['y'] < LA)]
```

Out[ ]:

	feature1	feature2	feature3	feature4	feature5	y
1012	639.532609	3.443599	4.493891	294.166114	40.952829	-330.935959
1816	656.670429	3.759442	4.554253	304.261278	45.020510	-285.299667

	feature1	feature2	feature3	feature4	feature5	y
2931	624.874864	4.406262	15.602420	241.443271	38.839708	370.538189
3140	592.095610	3.351936	11.833026	322.725064	36.138203	292.084397
3202	660.548305	3.396066	14.682695	231.652444	41.054453	286.945158
3353	627.315536	3.330178	4.759097	256.351126	44.375054	-301.524130
3527	628.209126	3.874566	8.372857	234.342040	44.686723	-295.395825
4028	665.318562	4.633223	12.881268	300.801012	45.768954	284.358748
4112	626.687103	3.666946	14.934568	222.582081	40.546789	303.192912
4340	658.583676	3.179204	14.510054	225.324214	31.829931	386.024044
4510	652.072945	3.671689	5.858203	236.460120	35.636061	-298.893225
4558	680.608545	4.203235	14.708863	268.216991	39.777674	294.176766
4584	630.525626	3.870085	2.726262	262.593083	46.343327	-305.811585
4690	643.667414	4.339807	1.914016	255.601645	47.952298	-329.716194
4827	622.483030	3.304024	13.876854	240.918547	38.875143	286.992836
5250	519.065777	3.811845	4.019342	245.116227	48.084591	-296.739444
5290	588.633760	3.568809	15.084998	180.006871	42.384177	288.958196
5337	685.115505	3.666659	15.312471	282.756486	57.962910	283.499562
5824	666.714973	3.412571	2.728916	261.053620	31.164544	-347.837176
6115	692.701297	3.276267	14.509550	234.811638	46.774550	292.831372
6562	637.560175	2.928713	3.618058	321.179099	61.114585	-297.248903
6716	634.198152	4.106935	5.247954	222.533840	47.267798	-296.878730
6912	609.712720	3.628976	4.612626	284.719937	42.455071	-282.770749
7172	596.176872	3.794074	14.334308	191.284769	42.561679	303.924191
7343	666.850513	3.517479	6.520315	197.364652	44.362522	-329.062651
7394	593.561243	3.357720	11.377143	349.052820	31.430776	296.064859
7554	648.002594	3.893116	3.852485	263.002021	35.176361	-300.149045
8189	619.658740	3.746888	4.097045	274.323809	43.668875	-326.334738
8359	637.137678	3.872600	3.972720	300.101618	42.673080	-332.100979
8419	572.876674	4.057061	15.041989	255.028675	29.437639	309.164448
8513	701.396741	3.933867	6.063132	308.624262	39.256262	-331.720135
8604	598.502454	3.517206	13.493327	198.522831	41.811966	289.825040
8668	619.918745	3.979028	13.112728	267.686243	53.467044	288.587766
8710	628.099316	3.660980	4.976833	222.099879	33.942306	-289.159069
8819	638.535290	3.747256	1.760734	259.675164	43.086146	-318.510328
8832	658.823226	3.814968	6.181304	233.839303	40.500524	-285.515143
9723	626.545245	2.816519	6.776332	268.257655	36.129799	-327.555629
9991	581.531883	3.479566	4.117059	206.940124	31.975190	-316.480725

In [ ]:
```
# trimming of Outliers
df_z = df_z[(df_z['y'] < HA) & (df_z['y'] > LA)]
df_z.head()
```

Out[ ]:

	feature1	feature2	feature3	feature4	feature5	y
0	845.112931	5.744720	18.839955	416.477928	82.153664	4.848880
1	799.264025	5.274422	16.702830	378.817945	72.949023	-13.091182
2	809.868188	5.383195	17.197115	387.528134	75.077918	7.353174
3	817.834662	5.464911	17.568451	394.071742	76.677270	-3.131487
4	792.396975	5.203983	16.382741	373.177397	71.570392	3.516674

In [ ]:
```
#capping on outliers
UL = df_z['y'].mean() + threshold*df_z['y'].std()
LL = df_z['y'].mean() - threshold*df_z['y'].std()
```

In [ ]:
```
#applying the capping

df_z['y'] = np.where(
```

```
    df_z['y']>UL, UL,
    np.where(
        df_z['y']<LL, LL,
        df_z['y']
    )
)
```

In [ ]:
```
# new statistics
df_z['y'].describe()
```

Out[ ]:
```
count    9962.000000
mean       -0.514235
std        92.034461
min      -276.636787
25%       -57.223631
50%        -2.893147
75%        57.742833
max       275.607769
Name: y, dtype: float64
```

2.5.2 IQR Based Filtering

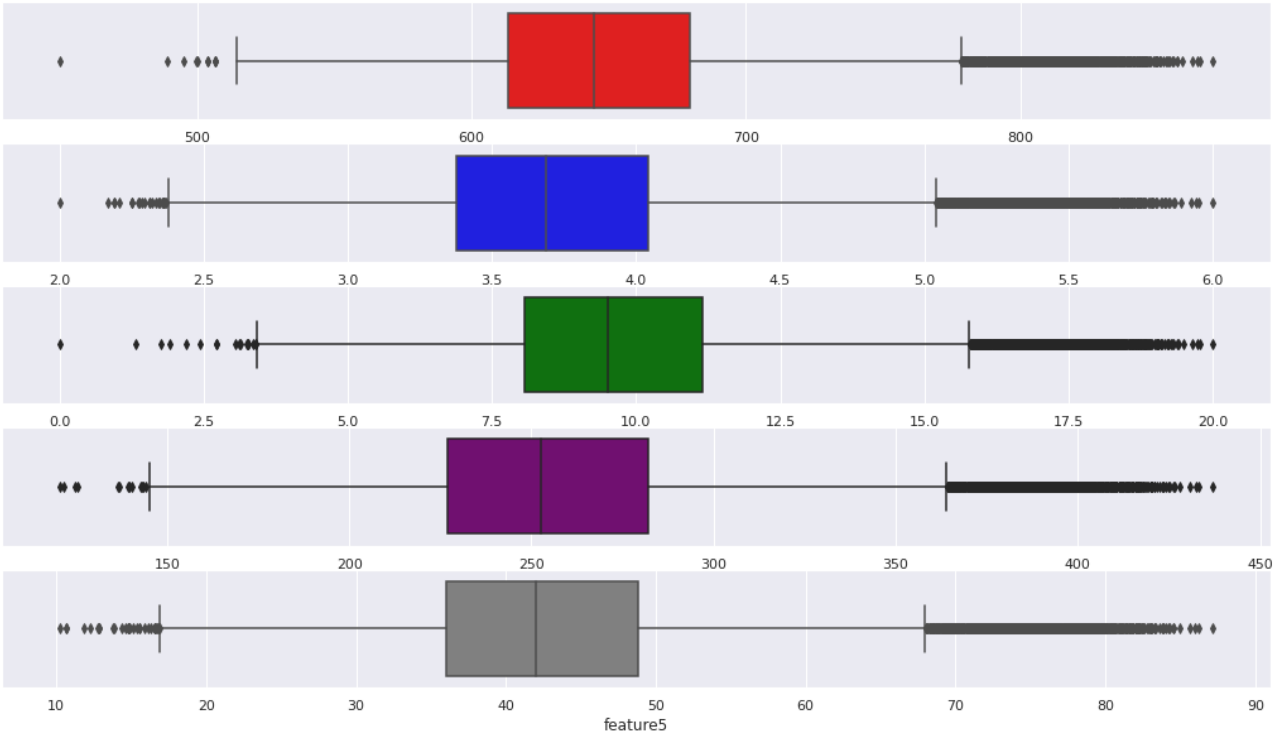
In [ ]:
```
# copy for safety reasons
df_iqr = df.copy()
```

In [ ]:
```
df_iqr.shape
```

Out[ ]:
```
(10000, 6)
```

In [ ]:
```
#boxplot for the skewed feature

plt.subplot(5,1,1)
sns.boxplot(df_iqr['feature1'], color ='red')

plt.subplot(5,1,2)
sns.boxplot(df_iqr['feature2'],color ='blue')

plt.subplot(5,1,3)
sns.boxplot(df_iqr['feature3'],color ='green')

plt.subplot(5,1,4)
sns.boxplot(df_iqr['feature4'],color ='purple')

plt.subplot(5,1,5)
sns.boxplot(df_iqr['feature5'],color ='gray')

#still outliers --> plotting before removal
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa5043350>
```



In [ ]:
```
#function to remove outliers using IQR

def remove_outlier_iqr(df, features):
```

```
  Q3 = np.quantile(df[features], 0.75)
  Q1 = np.quantile(df[features], 0.25)
  IQR = Q3 - Q1

  print("The IQR value for column %s is : %s" % (features, IQR))
  outlier_free_list = []
  global filtered_data

  lr = Q1 - 1.5 * IQR
  ur = Q3 + 1.5 * IQR

  print("The lr value for column %s is : %s" % (features, lr))
  print("The ur value for column %s is : %s" % (features, ur))

  for x in df[features]:
    if ((x >= lr) & (x <= ur)):
      outlier_free_list.append(x)

  filtered_data = df.loc[df[features].isin(outlier_free_list)]
  print(filtered_data.shape)
my_feats = ['feature1', 'feature2', 'feature3', 'feature4', 'feature5']
for i in df_iqr[my_feats]:
  remove_outlier_iqr(df_iqr, i)

df_iqr_new = filtered_data
```

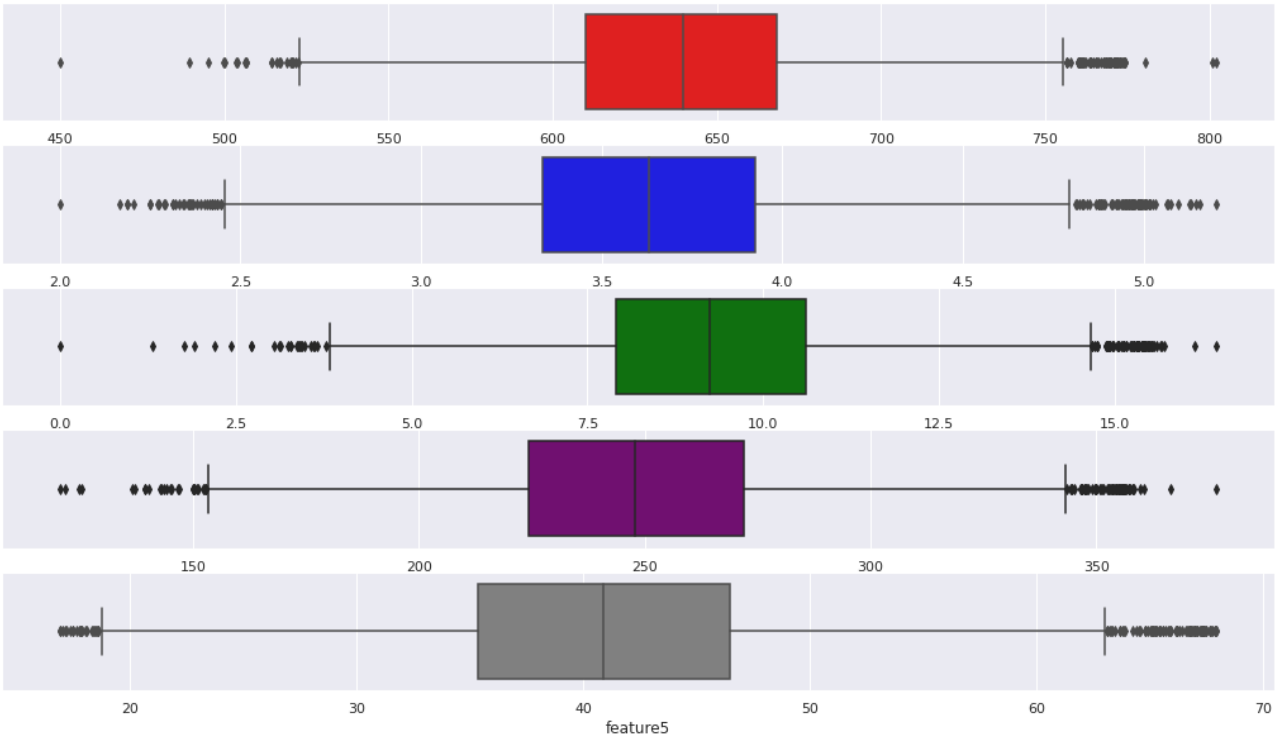
```
The IQR value for column feature1 is : 66.0687151749587
The lr value for column feature1 is : 514.2933836461845
The ur value for column feature1 is : 778.5682443460192
(9063, 6)
The IQR value for column feature2 is : 0.6672013340010992
The lr value for column feature2 is : 2.3732160029922014
The ur value for column feature2 is : 5.042021338996598
(9027, 6)
The IQR value for column feature3 is : 3.0919738510897403
The lr value for column feature3 is : 3.4231561838848883
The ur value for column feature3 is : 15.79105158824385
(9067, 6)
The IQR value for column feature4 is : 54.81715498559794
The lr value for column feature4 is : 144.85995106409885
The ur value for column feature4 is : 364.1285710064906
(9084, 6)
The IQR value for column feature5 is : 12.772565474960714
The lr value for column feature5 is : 16.869610790659415
The ur value for column feature5 is : 67.95987269050227
(9013, 6)
```

In [ ]:
```
df_iqr_new.shape
```

Out[ ]:
```
(9013, 6)
```

In [ ]:
```
plt.subplot(5,1,1)
sns.boxplot(df_iqr_new['feature1'], color ='red')

plt.subplot(5,1,2)
sns.boxplot(df_iqr_new['feature2'],color ='blue')

plt.subplot(5,1,3)
sns.boxplot(df_iqr_new['feature3'],color ='green')

plt.subplot(5,1,4)
sns.boxplot(df_iqr_new['feature4'],color ='purple')

plt.subplot(5,1,5)
sns.boxplot(df_iqr_new['feature5'],color ='gray')
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa4b7df50>
```

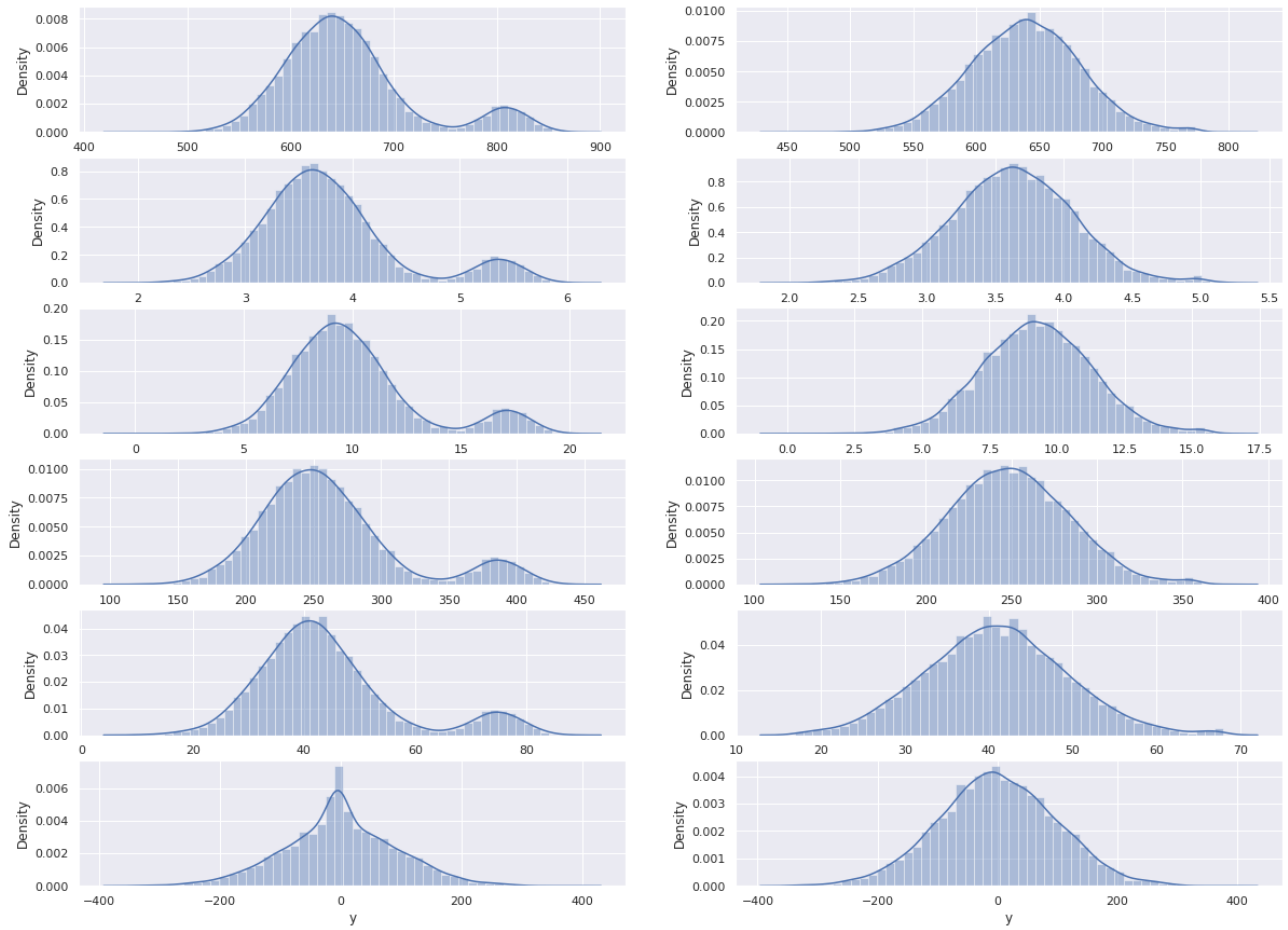


In [ ]:

```
#plots showind the outliers were removed

sns.set(rc={'figure.figsize':(20,15)})

plt.subplot(6,2,1)
sns.distplot(df_iqr['feature1'])

plt.subplot(6,2,2)
sns.distplot(df_iqr_new['feature1'])

plt.subplot(6,2,3)
sns.distplot(df_iqr['feature2'])

plt.subplot(6,2,4)
sns.distplot(df_iqr_new['feature2'])

plt.subplot(6,2,5)
sns.distplot(df_iqr['feature3'])

plt.subplot(6,2,6)
sns.distplot(df_iqr_new['feature3'])

plt.subplot(6,2,7)
sns.distplot(df_iqr['feature4'])

plt.subplot(6,2,8)
sns.distplot(df_iqr_new['feature4'])

plt.subplot(6,2,9)
sns.distplot(df_iqr['feature5'])

plt.subplot(6,2,10)
sns.distplot(df_iqr_new['feature5'])

plt.subplot(6,2,11)
sns.distplot(df_iqr['y'])

plt.subplot(6,2,12)
sns.distplot(df_iqr_new['y'])
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa0cd9210>
```



2.5.3 Percentile Based Filtering

In [ ]:
```
# copy for safety reasons
df_per = df.copy()
```

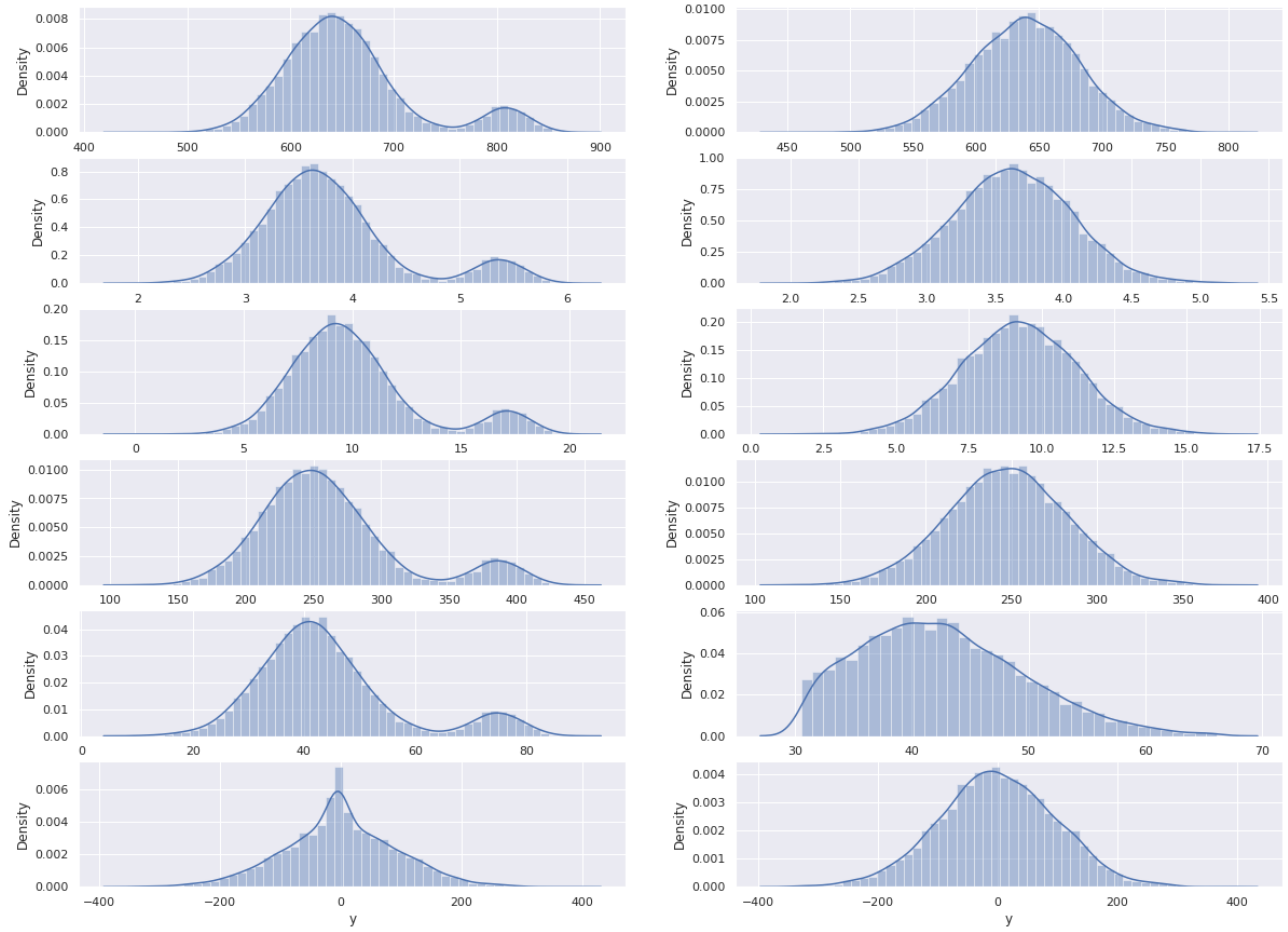
In [ ]:
```
#function to remove outliers using percentiles

def perc_remover(data, features):
  UL3 = data[features].quantile(0.90)
  LL3 = data[features].quantile(0.10)

  outlier_free_list = []
  global per_filtered_data

  print("The ll3 value for column %s is : %s" % (features, LL3))
  print("The ul3 value for column %s is : %s" % (features, UL3))

  for x in data[features]:
    if ((x > LL3) and (x < UL3)):
      outlier_free_list.append(x)

  per_filtered_data = data.loc[data[features].isin(outlier_free_list)]


for i in df_per[my_feats]:
  perc_remover(df_per, i)

new_per_df = per_filtered_data
```

```
The ll3 value for column feature1 is : 585.9929182246988
The ul3 value for column feature1 is : 762.5142474911396
The ll3 value for column feature2 is : 3.090141939334906
The ul3 value for column feature2 is : 4.941352402694089
The ll3 value for column feature3 is : 6.805996647943415
The ul3 value for column feature3 is : 15.088367240975987
The ll3 value for column feature4 is : 205.03210254367812
The ul3 value for column feature4 is : 350.0093934751661
The ll3 value for column feature5 is : 30.53232118993344
The ul3 value for column feature5 is : 66.13111462159546
```

In [ ]:
```
#plots showing outliers have been removed
sns.set(rc={'figure.figsize':(20,15)})

plt.subplot(6,2,1)
sns.distplot(df_per['feature1'])

plt.subplot(6,2,2)
sns.distplot(new_per_df['feature1'])
```

```
plt.subplot(6,2,3)
sns.distplot(df_per['feature2'])

plt.subplot(6,2,4)
sns.distplot(new_per_df['feature2'])

plt.subplot(6,2,5)
sns.distplot(df_per['feature3'])

plt.subplot(6,2,6)
sns.distplot(new_per_df['feature3'])

plt.subplot(6,2,7)
sns.distplot(df_per['feature4'])

plt.subplot(6,2,8)
sns.distplot(new_per_df['feature4'])

plt.subplot(6,2,9)
sns.distplot(df_per['feature5'])

plt.subplot(6,2,10)
sns.distplot(new_per_df['feature5'])

plt.subplot(6,2,11)
sns.distplot(df_per['y'])

plt.subplot(6,2,12)
sns.distplot(new_per_df['y'])
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8f9fd72e90>



2.5.4 Winsorization

In [ ]:
```
#copy for safety reasons
df_win = df.copy()
df_win_0 = df.copy()
```

In [ ]:
```
#function to remove outliers using winsorization

def win_func(data, features):
  UL3 = data[features].quantile(0.90)
  LL3 = data[features].quantile(0.10)

  print("The ll3 value for column %s is : %s" % (features, UL3))

  outlier_free_list = []
```

```
  global win_filtered_data

  for x in data[features]:
    if x <= LL3:
      data[features] = data[features].replace(x,LL3)
    elif x >= UL3:
      data[features] = data[features].replace(x,UL3)
    else:
      data[features]


    outlier_free_list.append(x)

  win_filtered_data = data.loc[data[features].isin(outlier_free_list)]


for i in df_win[my_feats]:
  win_func(df_win, i)

new_win_df = win_filtered_data
```

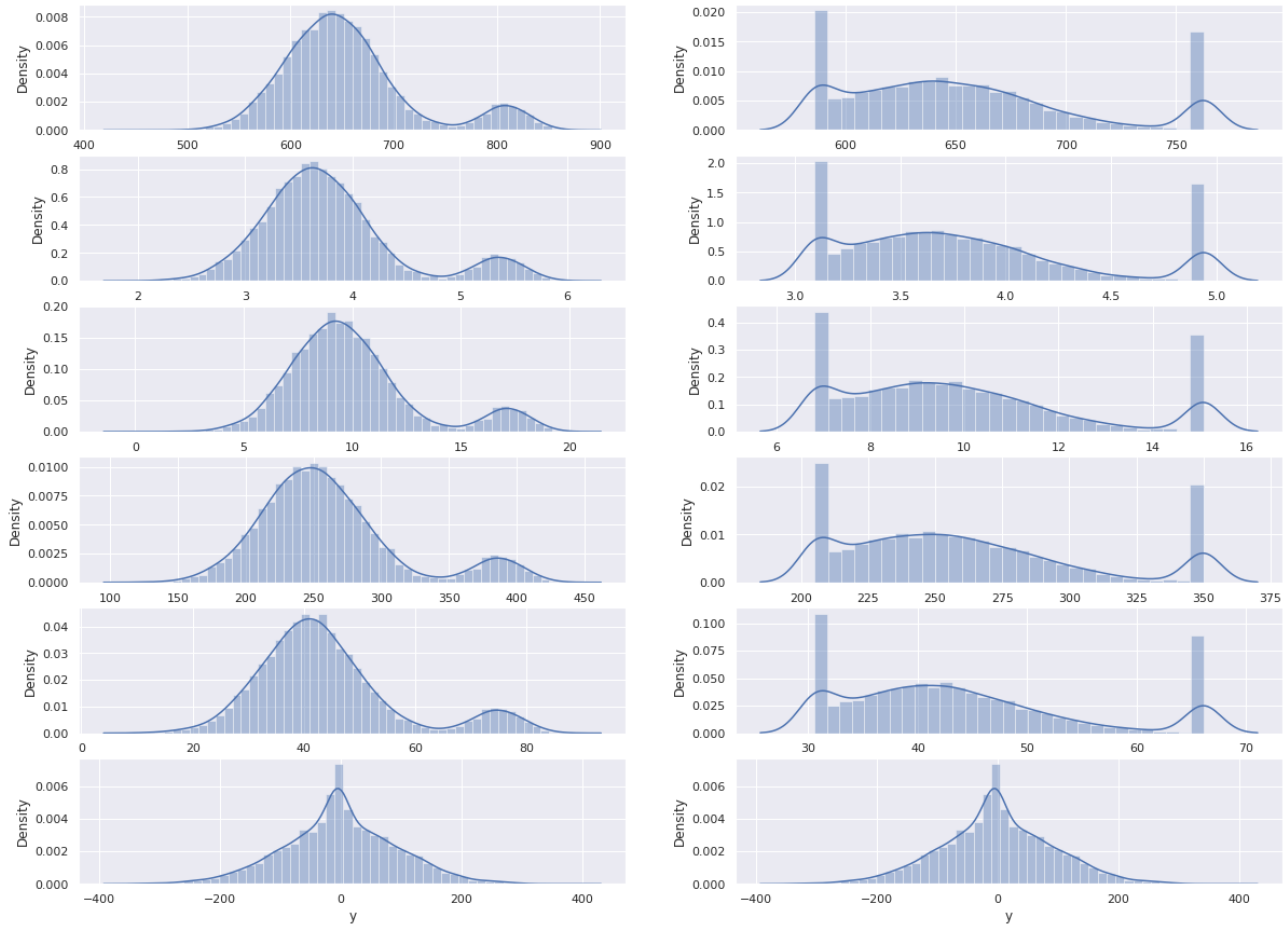
```
The ll3 value for column feature1 is : 762.4212486004184
The ll3 value for column feature2 is : 4.941237451778448
The ll3 value for column feature3 is : 15.085335232402326
The ll3 value for column feature4 is : 350.00107177730354
The ll3 value for column feature5 is : 66.12658953783802
```

In [ ]:
```
#apply capping (Winsorization)
new_win_df.shape
```

Out[ ]:
```
(8000, 6)
```

In [ ]:
```
#plots showing outliers have been removed

sns.set(rc={'figure.figsize':(20,15)})

plt.subplot(6,2,1)
sns.distplot(df_win_0['feature1'])

plt.subplot(6,2,2)
sns.distplot(new_win_df['feature1'])

plt.subplot(6,2,3)
sns.distplot(df_win_0['feature2'])

plt.subplot(6,2,4)
sns.distplot(new_win_df['feature2'])

plt.subplot(6,2,5)
sns.distplot(df_win_0['feature3'])

plt.subplot(6,2,6)
sns.distplot(new_win_df['feature3'])

plt.subplot(6,2,7)
sns.distplot(df_win_0['feature4'])

plt.subplot(6,2,8)
sns.distplot(new_win_df['feature4'])

plt.subplot(6,2,9)
sns.distplot(df_win_0['feature5'])

plt.subplot(6,2,10)
sns.distplot(new_win_df['feature5'])

plt.subplot(6,2,11)
sns.distplot(df_win_0['y'])

plt.subplot(6,2,12)
sns.distplot(new_win_df['y'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8f9ecc5f90>
```



2.6 Methods and Models - Synthetic Dataset

Despite having completed 4 methods of detecting and removing outliers, in this section where different regression models are used to compare the original dataset and the dataset with the outliers removed, we will just be using the datasets that used the IQR Based Filtering method of detecting and removing outliers

2.6.1 Linear Regression

In [ ]:
```
#Load the synthetic dataset
X.shape
```

Out[ ]:
```
(10000, 5)
```

In [ ]:
```
#split the data into training and test sets
X_train = X[:-2000]
X_test = X[-2000:]
```

In [ ]:
```
#split the target into training and test sets
y_train = y[:-2000]
y_test = y[-2000:]
```

In [ ]:
```
y_train.shape
```

Out[ ]:
```
(8000,)
```

In [ ]:
```
#create linear regression object
from sklearn import linear_model
regr = linear_model.LinearRegression()
```

In [ ]:
```
#train the model using the training sets (original dataset)
regr.fit(X_train, y_train)
```

Out[ ]:
```
LinearRegression()
```

In [ ]:
```
#make predictions using the test set
y_pred = regr.predict(X_test)
```

In [ ]:
```
#get the coefficients, beta
regr.coef_
```

Out[ ]:
```
array([-13.7640045 , -13.69269491,  59.9916997 , -13.13143256,
       -13.70109408])
```

In [ ]:
```
#getting the intercept, alpha
regr.intercept_
```

Out[ ]:
```
-4.12826618431209
```

In [ ]:
```
from sklearn.metrics import mean_squared_error, r2_score
# coefficients
print('Coefficients: \n', regr.coef_)
# mean squared error
print('Mean squared error: %.2f'% mean_squared_error(y_pred, y_test))
# coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(y_pred, y_test))
```

```
Coefficients:
 [-13.7640045  -13.69269491  59.9916997  -13.13143256 -13.70109408]
Mean squared error: 5491.18
Coefficient of determination: -0.24
```

Linear Regression - On dataset after IQR removal method

In [ ]:
```
IQR_reg = linear_model.LinearRegression()
IQR_reg.fit(df_iqr[['feature1','feature2','feature3','feature4', 'feature5']], df_iqr['y'])
```

Out[ ]:
```
LinearRegression()
```

In [ ]:
```
IQR_pred = IQR_reg.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import mean_squared_error, r2_score
# coefficients
print('Coefficients: \n', IQR_reg.coef_)
# mean squared error
print('Mean squared error: %.2f'% mean_squared_error(IQR_pred, y_test))
# coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(IQR_pred, y_test))
```

```
Coefficients:
 [ -0.32138211 -29.88886696  30.58494077  -0.37395888  -1.59481198]
Mean squared error: 41650.89
Coefficient of determination: -22.34
```

2.6.2 XGBoost Regressor

In [ ]:
```
XGBR = XGBRegressor()
#train the model using the training sets
XGBR.fit(X_train, y_train)
```

```
[18:46:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

Out[ ]:
```
XGBRegressor()
```

In [ ]:
```
#make predictions using the test set
y_predXG = XGBR.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import mean_squared_error, r2_score
# mean squared error
print('Mean squared error: %.2f'% mean_squared_error(y_predXG, y_test))
# coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(y_predXG, y_test))
```

```
Mean squared error: 4481.49
Coefficient of determination: 0.17
```

XGBoost Regressor - On Dataset after IQR removal method

In [ ]:
```
IQR_XGBR = XGBRegressor()
IQR_XGBR.fit(df_iqr[['feature1','feature2','feature3','feature4', 'feature5']].values, df_iqr['y'].values)
```

```
[18:46:26] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

Out[ ]:
```
XGBRegressor()
```

In [ ]:
```
IQR_pred2 = IQR_XGBR.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import mean_squared_error, r2_score
# mean squared error
print('Mean squared error: %.2f'% mean_squared_error(IQR_pred2, y_test))
# coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(IQR_pred2, y_test))
```

```
Mean squared error: 66628.93
Coefficient of determination: -9549.92
```

2.6.3 KNeighbours - As a Regressor

In [ ]:
```
from sklearn.neighbors import KNeighborsRegressor
K = 3
neigh = KNeighborsRegressor(n_neighbors=K)
neigh.fit(X_train, y_train)
```

Out[ ]:
```
KNeighborsRegressor(n_neighbors=3)
```

In [ ]:
```
print(neigh.predict(X_test))
```

```
[  11.16783312 -167.09057794   43.91566126 ...   71.70377114  -93.52564315
  128.93736786]
```

In [ ]:
```
#making predictions using the test set
y_predKNR = neigh.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import mean_squared_error, r2_score
# mean squared error
print('Mean squared error: %.2f'% mean_squared_error(y_predKNR, y_test))
# coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(y_predKNR, y_test))
```

```
Mean squared error: 6074.27
Coefficient of determination: -0.01
```

KNeighbours Regressor - On Dataset after IQR removal method

In [ ]:
```
IQR_neigh = KNeighborsRegressor(n_neighbors=K)
IQR_neigh.fit(df_iqr[['feature1','feature2','feature3','feature4', 'feature5']], df_iqr['y'])
```

Out[ ]:
```
KNeighborsRegressor(n_neighbors=3)
```

In [ ]:
```
IQR_pred3 = IQR_neigh.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import mean_squared_error, r2_score
# mean squared error
print('Mean squared error: %.2f'% mean_squared_error(IQR_pred3, y_test))
# coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(IQR_pred3, y_test))
```

```
Mean squared error: 15764.40
Coefficient of determination: -78061506214237193512615940194304.00
```

2.6.4 Logistic Regression

In [ ]:
```
lr_o = LogisticRegression()
lr_o.fit(X_train, np.floor(y_train))
```

Out[ ]:
```
LogisticRegression()
```

In [ ]:
```
lr_pred = lr_o.predict(X_test)
```

In [ ]:
```
print('Mean squared error: %.2f'% mean_squared_error(lr_pred, np.floor(y_test)))
# coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(lr_pred, np.floor(y_test)))
```

```
Mean squared error: 6442.58
Coefficient of determination: -0.17
```

Logistic Regression - On Dataset after IQR removal method

In [ ]:
```
IQR_lr = LogisticRegression()
IQR_lr.fit(df_iqr[['feature1','feature2','feature3','feature4', 'feature5']], np.floor(df_iqr['y']))
```

Out[ ]:
```
LogisticRegression()
```

In [ ]:
```
IQR_pred4 = IQR_lr.predict(X_test)
```

In [ ]:
```
print('Mean squared error: %.2f'% mean_squared_error(IQR_pred4, np.floor(y_test)))
# coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(IQR_pred4, np.floor(y_test)))
```

```
Mean squared error: 14021.12
Coefficient of determination: -0.32
```

3. California Housing Dataset

3.1 Data Preparation

In [ ]:
```
#Importing Dataset from Sklearn
california_housing = fetch_california_housing(as_frame=True)
```

In [ ]:
```
#Printing the Description of the Data
print(california_housing.DESCR)
```

```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the target

    :Attribute Information:
        - MedInc        median income in block group
        - HouseAge      median house age in block group
        - AveRooms      average number of rooms per household
        - AveBedrms     average number of bedrooms per household
        - Population    block group population
        - AveOccup      average number of household members
        - Latitude      block group latitude
        - Longitude     block group longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

The target variable is the median house value for California districts,
expressed in hundreds of thousands of dollars ($100,000).

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S.
Census Bureau publishes sample data (a block group typically has a population
of 600 to 3,000 people).

An household is a group of people residing within a home. Since the average
number of rooms and bedrooms in this dataset are provided per household, these
columns may take surpinsingly large values for block groups with few households
and many empty houses, such as vacation resorts.

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. topic:: References

    - Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
      Statistics and Probability Letters, 33 (1997) 291-297
```

In [ ]:
```
#Overview of the Dataset
california_housing.frame.head()
```

Out[ ]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

In [ ]:
```
features = ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup',]
```

In [ ]:
```
#Overview of the Target Data
california_housing.target.head()
```

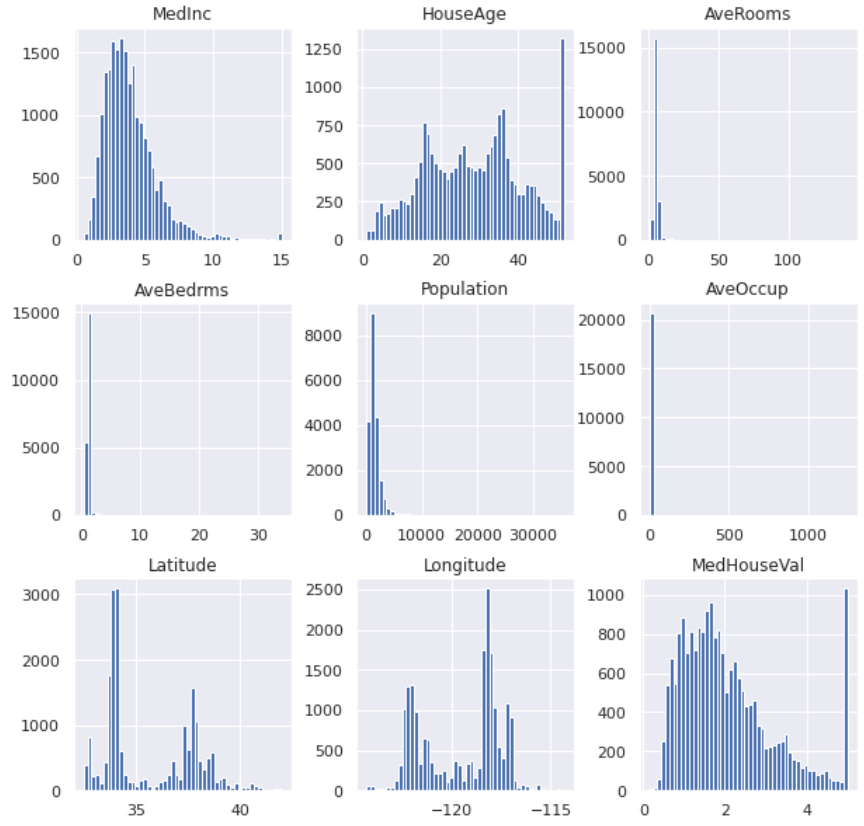
Out[ ]:
```
0    4.526
1    3.585
2    3.521
3    3.413
4    3.422
Name: MedHouseVal, dtype: float64
```

In [ ]:
```
#Checking the Data Types and Null Values
california_housing.frame.info()
```

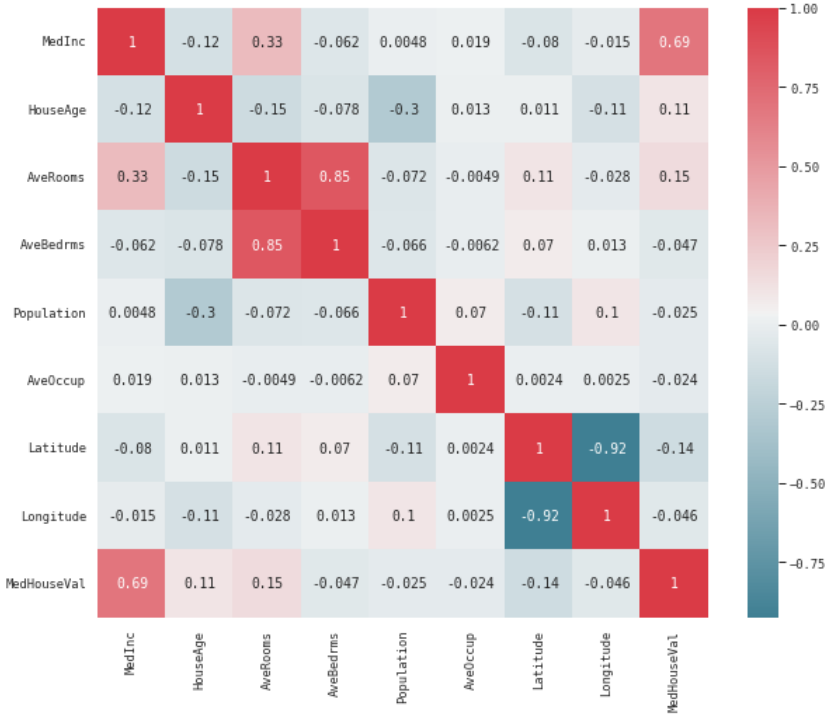
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   MedInc       20640 non-null  float64
 1   HouseAge     20640 non-null  float64
 2   AveRooms     20640 non-null  float64
 3   AveBedrms    20640 non-null  float64
 4   Population   20640 non-null  float64
 5   AveOccup     20640 non-null  float64
 6   Latitude     20640 non-null  float64
 7   Longitude    20640 non-null  float64
 8   MedHouseVal  20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

3.2 Data Visualisation

In [ ]:
```
#Plotting data on Histograms
california_housing.frame.hist(figsize=(10,10),bins=50);
```

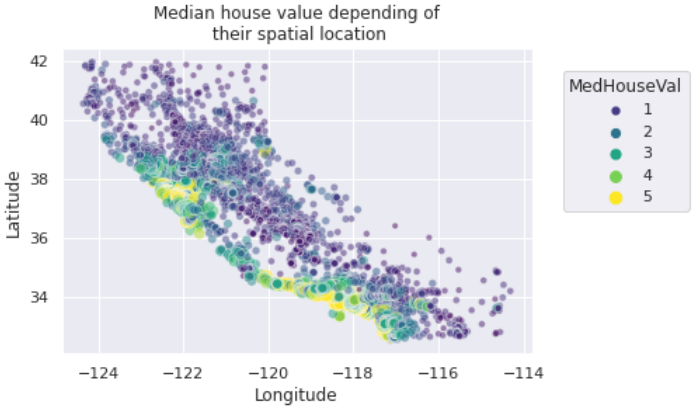


In [ ]:
```
#Plotting data on a Correlation Matrix
#set the context for plotting
sns.set(context="paper",font="monospace")
housing_corr_matrix = california_housing.frame.corr()
#set the matplotlib figure
fig, axe = plt.subplots(figsize=(10,8))
#Generate color palettes
cmap = sns.diverging_palette(220,10,center = "light", as_cmap=True)
#draw the heatmap
sns.heatmap(housing_corr_matrix,vmax=1,square =True, cmap=cmap,annot=True );
```



In [ ]:

```
#Scatterplot of Latitutde, Longitude and Median House Value
sns.set(rc = {'figure.figsize':(6,4)})

sns.scatterplot(data=california_housing.frame, x="Longitude", y="Latitude",
                size="MedHouseVal", hue="MedHouseVal",
                palette="viridis", alpha=0.5)
plt.legend(title="MedHouseVal", bbox_to_anchor=(1.05, 0.95),
           loc="upper left")
_ = plt.title("Median house value depending of\n their spatial location")
```



In [ ]:

```
#Random Subsampling to have less data points to plot
rng = np.random.RandomState(0)
indices = rng.choice(np.arange(california_housing.frame.shape[0]), size=1000,
                     replace=False)
```

In [ ]:

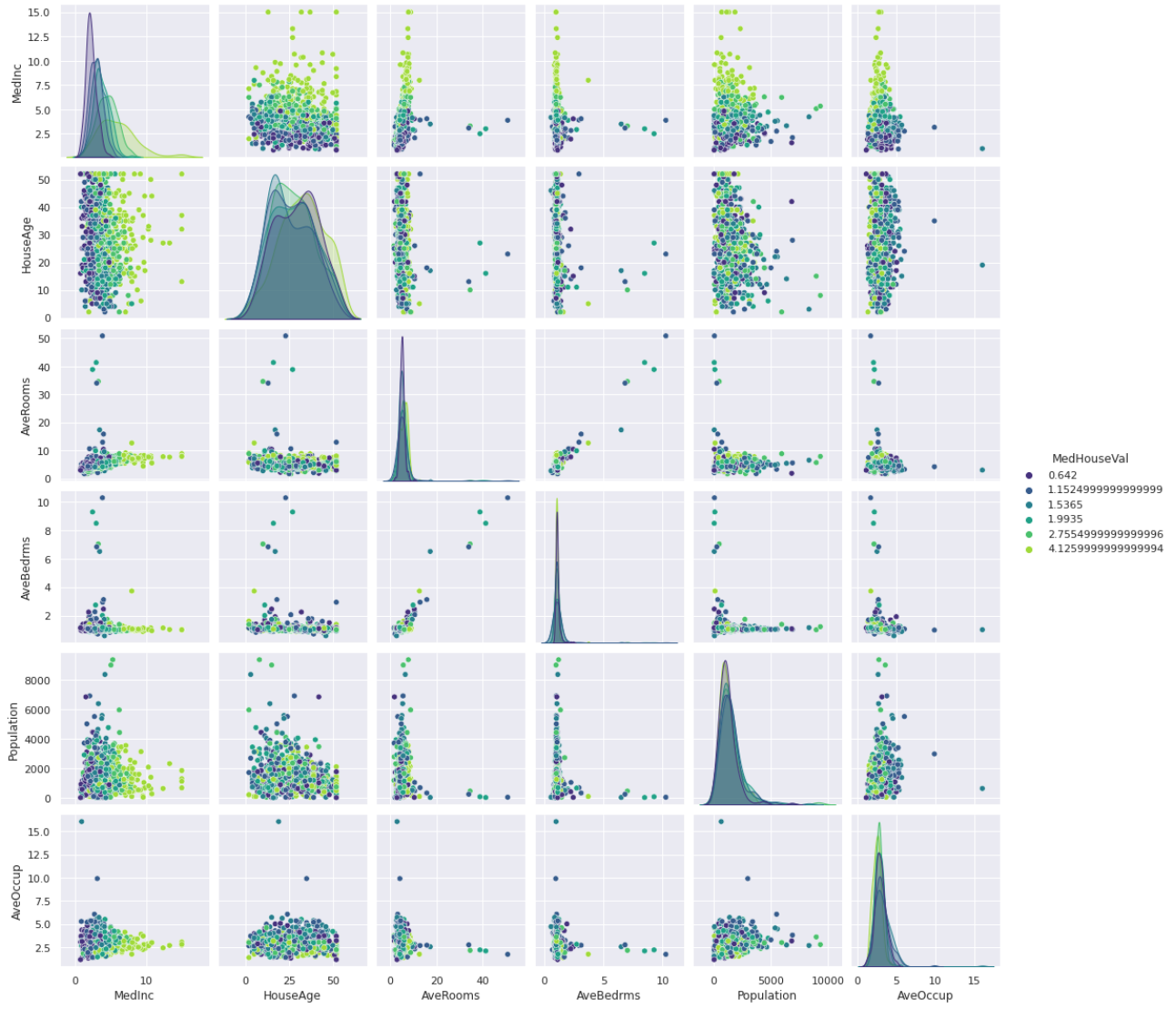
```
# Drop the unwanted columns
columns_drop = ["Longitude", "Latitude"]
subset = california_housing.frame.iloc[indices].drop(columns=columns_drop)
# Quantize the target and keep the midpoint for each interval
subset["MedHouseVal"] = pd.qcut(subset["MedHouseVal"], 6, retbins=False)
subset["MedHouseVal"] = subset["MedHouseVal"].apply(lambda x: x.mid)
```

In [ ]:

```
#Scatterplots of the features
sns.set(rc = {'figure.figsize':(6,4)})

sns.pairplot(data=subset, hue="MedHouseVal", palette="viridis")
```

Out[ ]:

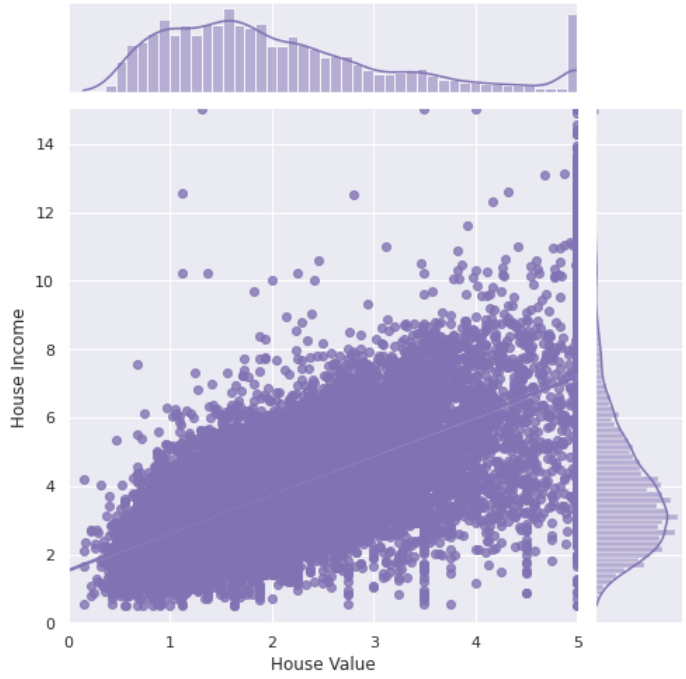
```
<seaborn.axisgrid.PairGrid at 0x7f8faa9ad910>
```



In [ ]:

```
#Scatterplot with Distribution of House Value & House Income
sns.set_theme(style="darkgrid")

Value = california_housing.frame
g = sns.jointplot(x="MedHouseVal", y="MedInc",  data=Value,
                  kind="reg", truncate=False,
                  xlim=(0, 5), ylim=(0, 15),
                  color="m", height=7)
g.set_axis_labels('House Value', 'House Income')
```

Out[ ]: <seaborn.axisgrid.JointGrid at 0x7f8fa827f590>



3.3 Detecting NANs

In [ ]:
```
#Checking for null values
california_housing.frame.isnull().sum()
```

Out[ ]:
```
MedInc          0
HouseAge        0
AveRooms        0
AveBedrms       0
Population      0
AveOccup        0
Latitude        0
Longitude       0
MedHouseVal     0
dtype: int64
```

3.4 Training test split

In [ ]:
```
#Copying and Splitting Data
california_housing.frame_Z = california_housing.frame.copy()
X2_Z = california_housing.frame_Z.drop("MedHouseVal",axis=1)
y2_Z = california_housing.frame_Z["MedHouseVal"]
```

In [ ]:
```
#Copying and Splitting Data
california_housing.frame_IQR = california_housing.frame.copy()
X2_IQR = california_housing.frame_IQR.drop("MedHouseVal",axis=1)
y2_IQR = california_housing.frame_IQR["MedHouseVal"]
```

In [ ]:
```
#Copying and Splitting Data
california_housing.frame_P = california_housing.frame.copy()
X2_P = california_housing.frame_P.drop("MedHouseVal",axis=1)
y2_P = california_housing.frame_P["MedHouseVal"]
```

In [ ]:
```
#Copying and Splitting Data
california_housing.frame_W = california_housing.frame.copy()
X2_W = california_housing.frame_W.drop("MedHouseVal",axis=1)
y2_W = california_housing.frame_W["MedHouseVal"]
```

3.5 Outlier Techniques

3.5.1 Z-Score

In [ ]:
```
#Splitting the Data
X2_train, X2_test, y2_train, y2_test = train_test_split(X2_Z,y2_Z,test_size=0.2,random_state=42)
print("X_train shape {} and size {}".format(X2_train.shape,X2_train.size))
print("X_test shape {} and size {}".format(X2_test.shape,X2_test.size))
print("y_train shape {} and size {}".format(y2_train.shape,y2_train.size))
print("y_test shape {} and size {}".format(y2_test.shape,y2_test.size))
```

```
X_train shape (16512, 8) and size 132096
X_test shape (4128, 8) and size 33024
```

```
y_train shape (16512,) and size 16512
y_test shape (4128,) and size 4128
```

Linear Regression

In [ ]:
```
#Scaling the data
independent_scaler = StandardScaler()
X2_train_scal = independent_scaler.fit_transform(X2_train)
X2_test_scal = independent_scaler.transform(X2_test)
print("Train Data: ")
print(X2_train_scal[0:1,0:])
print("Test Data: ")
print(X2_test_scal[0:1,0:])
```

```
Train Data:
[[-0.326196    0.34849025 -0.17491646 -0.20836543  0.76827628  0.05137609
  -1.3728112   1.27258656]]
Test Data:
[[-1.15508475 -0.28632369 -0.52068576 -0.17174603 -0.03030109  0.06740798
   0.1951      0.28534728]]
```

In [ ]:
```
#Linear Regression
LinRegModel = LinearRegression(n_jobs=-1)
#Fit the model to the training data (learn the coefficients)
LinRegModel.fit(X2_train_scal,y2_train)
#Print the intercept and coefficients
print("Intercept: "+str(LinRegModel.intercept_))
print("Coefficients: "+str(LinRegModel.coef_))
```

```
Intercept: 2.0719469373788777
Coefficients: [ 0.85438303  0.12254624 -0.29441013  0.33925949 -0.00230772 -0.0408291
 -0.89692888 -0.86984178]
```

In [ ]:
```
#Prediction on Test Data
y2_pred = LinRegModel.predict(X2_test_scal)
#Predictions and Test Values
print("Length of y2 Predictions:", len(y2_pred))
print("Length of y2 Tests:", len(y2_test))
test = pd.DataFrame({'Predicted':y2_pred,'Actual':y2_test})
test.head()
```

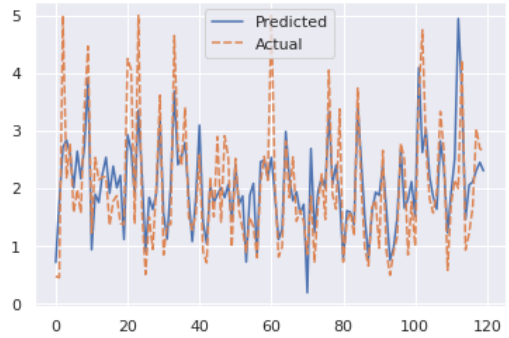
```
Length of y2 Predictions: 4128
Length of y2 Tests: 4128
```

Out[ ]:

	Predicted	Actual
20046	0.719123	0.47700
3024	1.764017	0.45800
15663	2.709659	5.00001
20484	2.838926	2.18600
9814	2.604657	2.78000

In [ ]:
```
#Charting the Predictions vs Actual
fig= plt.figure(figsize=(6,4))
test = test.reset_index()
test = test.drop(['index'],axis=1)
sns.lineplot(data=test[:120], sort=False)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa81727d0>
```



Logstic Regression

In [ ]:
```
#Creating and Fitting the Model
Logreg2 = LogisticRegression(solver='lbfgs', max_iter=10000)
Logreg2.fit(X2_train, np.ceil(y2_train))
```

Out[ ]:
```
LogisticRegression(max_iter=10000)
```

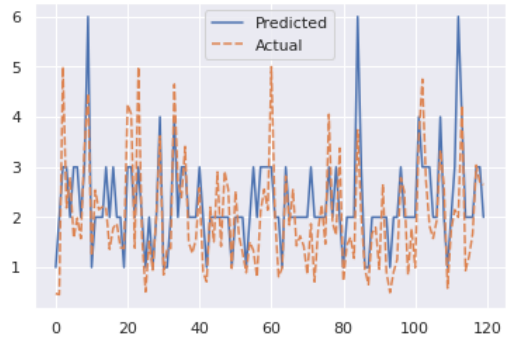
In [ ]:
```
#Creating and Printing Scores
```

```
y2_pred = Logreg2.predict(X2_test)
print('Accuracy: %.3f' % accuracy_score(np.ceil(y2_test), y2_pred))
print('Precision: %.3f' % precision_score(np.ceil(y2_test), y2_pred, average="weighted"))
print('Recall: %.3f' % recall_score(np.ceil(y2_test), y2_pred, average="weighted"))
print('F-measure: %.3f' % f1_score(np.ceil(y2_test), y2_pred, average="weighted"))
```

```
Accuracy: 0.568
Precision: 0.535
Recall: 0.568
F-measure: 0.539
```

In [ ]:
```
#Graphically representing Actual vs Predicted
LogGraph = pd.DataFrame({'Predicted':y2_pred, 'Actual': y2_test})
LogGraph = LogGraph.reset_index()
LogGraph = LogGraph.drop(['index'], axis=1)

fig= plt.figure(figsize=(6,4))
sns.lineplot(data=LogGraph[:120])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa7e8c990>
```



Kneighbours

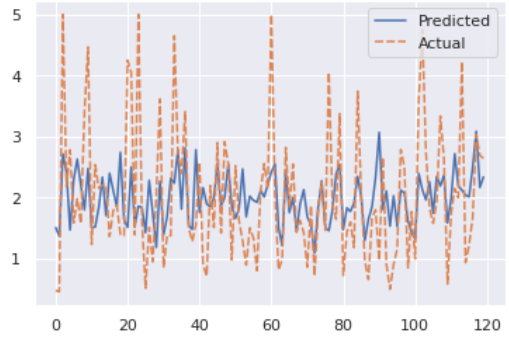
In [ ]:
```
#KNeighbors Regression
knn_regressor = KNeighborsRegressor(n_neighbors = 7).fit(X2_train, y2_train)
y2_pred = knn_regressor.predict(X2_test)
```

In [ ]:
```
#KNeighbours Training & Testing Scores
print("Training score : ", knn_regressor.score(X2_train, y2_train))
print("Testing score : ", r2_score(y2_test, y2_pred))
```

```
Training score :  0.38498374447845496
Testing score :  0.1571042759473391
```

In [ ]:
```
#Graphically representing Actual vs Predicted
KGraph = pd.DataFrame({'Predicted':y2_pred, 'Actual': y2_test})
KGraph = KGraph.reset_index()
KGraph = KGraph.drop(['index'], axis=1)

fig= plt.figure(figsize=(6,4))
sns.lineplot(data=KGraph[:120])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa7e20610>
```



XGBRegressor

In [ ]:
```
#XGBRegressor
xg_reg = xgb.XGBRegressor(objective ='reg:squarederror', colsample_bytree = 1,eta=0.3, learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 2000)
```

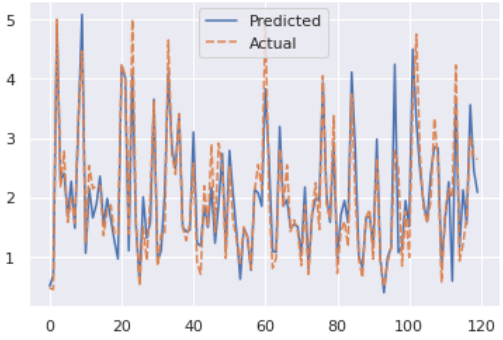
In [ ]:
```
#Fitting & Predicting the Model
xg_reg.fit(X2_train,y2_train)
y2_pred = xg_reg.predict(X2_test)
```

In [ ]:
```
#Producing R2
r2xgb=r2_score(y2_test,y2_pred)
print('the R squared of the xgboost method is:', r2xgb)
```

```
the R squared of the xgboost method is: 0.8461485016011139
```

In [ ]:
```
#Graphically representing Actual vs Predicted
grp = pd.DataFrame({'Predicted':y2_pred,'Actual':y2_test})
grp = grp.reset_index()
grp = grp.drop(['index'],axis=1)
fig= plt.figure(figsize=(6,4))
sns.lineplot(data=grp[:120])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa7ddcc90>
```



In [ ]:
```
#Splitting the Data
X2_train, X2_test, y2_train, y2_test = train_test_split(X2_IQR,y2_IQR,test_size=0.2,random_state=42)
print("X_train shape {} and size {}".format(X2_train.shape,X2_train.size))
print("X_test shape {} and size {}".format(X2_test.shape,X2_test.size))
print("y_train shape {} and size {}".format(y2_train.shape,y2_train.size))
print("y_test shape {} and size {}".format(y2_test.shape,y2_test.size))
```

```
X_train shape (16512, 8) and size 132096
X_test shape (4128, 8) and size 33024
y_train shape (16512,) and size 16512
y_test shape (4128,) and size 4128
```

3.5.2 IQR Based Filtering

In [ ]:
```
#Splitting the Data
X2_train, X2_test, y2_train, y2_test = train_test_split(X2_IQR,y2_IQR,test_size=0.2,random_state=42)
print("X_train shape {} and size {}".format(X2_train.shape,X2_train.size))
print("X_test shape {} and size {}".format(X2_test.shape,X2_test.size))
print("y_train shape {} and size {}".format(y2_train.shape,y2_train.size))
print("y_test shape {} and size {}".format(y2_test.shape,y2_test.size))
```

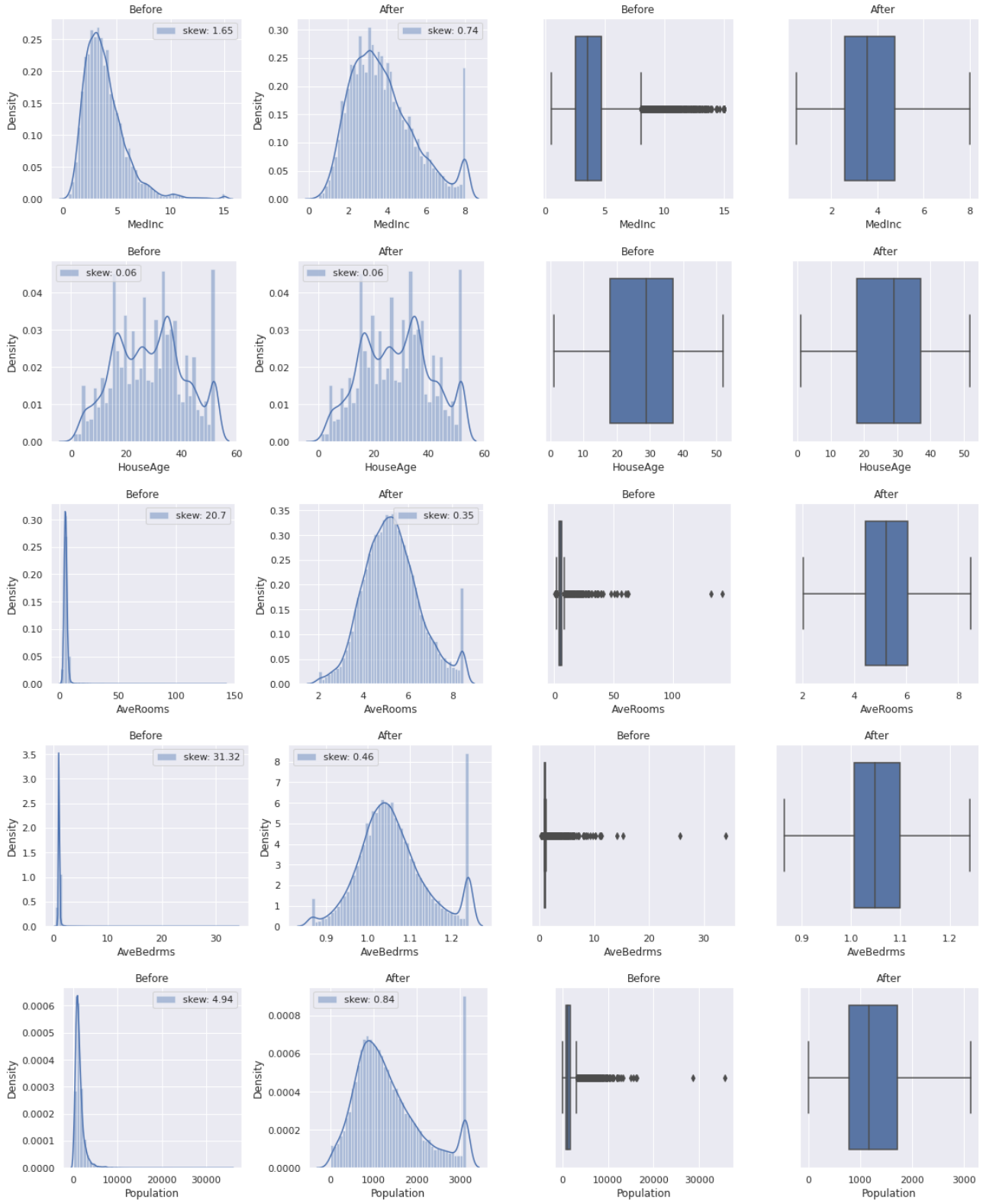
```
X_train shape (16512, 8) and size 132096
X_test shape (4128, 8) and size 33024
y_train shape (16512,) and size 16512
y_test shape (4128,) and size 4128
```

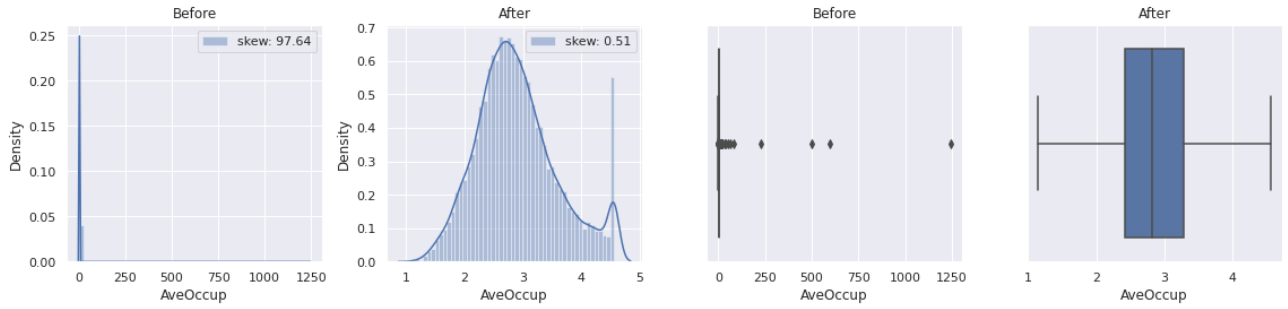
In [ ]:
```
def iqr_capping(df, cols, factor):

    for col in cols:

        q1 = california_housing.frame_IQR[col].quantile(0.25)
        q3 = california_housing.frame_IQR[col].quantile(0.75)

        iqr = q3 - q1

        upper_whisker = q3 + (factor*iqr)
        lower_whisker = q1 - (factor*iqr)

        df[col] = np.where(df[col]>upper_whisker, upper_whisker,
                 np.where(df[col]<lower_whisker, lower_whisker, df[col]))
```

In [ ]:
```
iqr_capping(california_housing.frame_IQR, features, 1.5)
```

In [ ]:
```
for col in features:
    plt.figure(figsize=(16,4))

    plt.subplot(141)
    sns.distplot(california_housing.frame[col], label="skew: " + str(np.round(california_housing.frame[col].skew(),2)))
    plt.title('Before')
    plt.legend()

    plt.subplot(142)
    sns.distplot(california_housing.frame_IQR[col], label="skew: " + str(np.round(california_housing.frame_IQR[col].skew(),2)))
    plt.title('After')
    plt.legend()
```

```
    plt.subplot(143)
    sns.boxplot(california_housing.frame[col])
    plt.title('Before')

    plt.subplot(144)
    sns.boxplot(california_housing.frame_IQR[col])
    plt.title('After')
    plt.tight_layout()
    plt.show()
```





Linear Regression

In [ ]:
```
#Linear Model
LinRegModel = LinearRegression(n_jobs=-1)
#Fit the model to the training data (learn the coefficients)
LinRegModel.fit(X2_train,y2_train)
#Print the intercept and coefficients
print("Intercept: "+str(LinRegModel.intercept_))
print("Coefficients: "+str(LinRegModel.coef_))
```

```
Intercept: -37.02327770606412
Coefficients: [ 4.48674910e-01  9.72425752e-03 -1.23323343e-01  7.83144907e-01
 -2.02962058e-06 -3.52631849e-03 -4.19792487e-01 -4.33708065e-01]
```

In [ ]:
```
#Prediction on Test Data
y2_pred = LinRegModel.predict(X2_test)

#Predictions and Test Values
print("Length of y2 Predictions:", len(y2_pred))
print("Length of y2 Tests:", len(y2_test))

test = pd.DataFrame({'Predicted':y2_pred,'Actual':y2_test})
test.head()
```

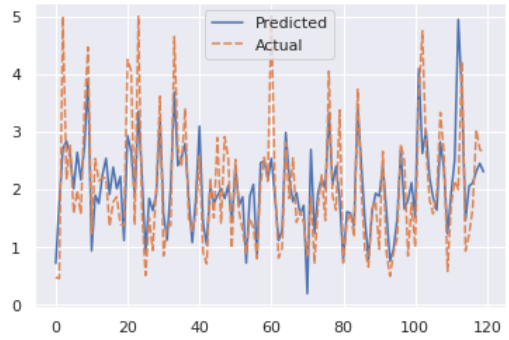
```
Length of y2 Predictions: 4128
Length of y2 Tests: 4128
```

Out[ ]:

	Predicted	Actual
20046	0.719123	0.47700
3024	1.764017	0.45800
15663	2.709659	5.00001
20484	2.838926	2.18600
9814	2.604657	2.78000

In [ ]:
```
#Charting the Predictions vs Actual
fig= plt.figure(figsize=(6,4))
test = test.reset_index()
test = test.drop(['index'],axis=1)
sns.lineplot(data=test[:120], sort=False)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa7479290>
```



Logstic Regression

In [ ]:
```
#Logistic Regression
Logreg2 = LogisticRegression(solver='lbfgs', max_iter=10000)
Logreg2.fit(X2_train, np.ceil(y2_train))
```

Out[ ]:
```
LogisticRegression(max_iter=10000)
```

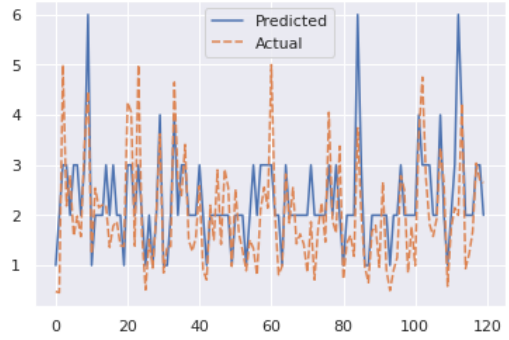
In [ ]:
```
y2_pred = Logreg2.predict(X2_test)
print('Accuracy: %.3f' % accuracy_score(np.ceil(y2_test), y2_pred))
print('Precision: %.3f' % precision_score(np.ceil(y2_test), y2_pred, average="weighted"))
```

```
print('Recall: %.3f' % recall_score(np.ceil(y2_test), y2_pred, average="weighted"))
print('F-measure: %.3f' % f1_score(np.ceil(y2_test), y2_pred, average="weighted"))
```

```
Accuracy: 0.568
Precision: 0.535
Recall: 0.568
F-measure: 0.539
```

In [ ]:
```
#Graphically representing Actual vs Predicted
LogGraph = pd.DataFrame({'Predicted':y2_pred, 'Actual': y2_test})
LogGraph = LogGraph.reset_index()
LogGraph = LogGraph.drop(['index'], axis=1)

fig= plt.figure(figsize=(6,4))
sns.lineplot(data=LogGraph[:120])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa8405a10>
```



Kneighbours

In [ ]:
```
#KNeighbors Regression
knn_regressor = KNeighborsRegressor(n_neighbors = 10).fit(X2_train, y2_train)
```

In [ ]:
```
print("Training score : ", knn_regressor.score(X2_train, y2_train))
```

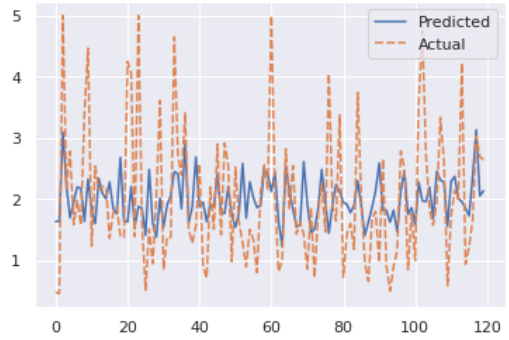
```
Training score :  0.3216293223790314
```

In [ ]:
```
y2_pred = knn_regressor.predict(X2_test)
print("testing score : ", r2_score(y2_test, y2_pred))
```

```
testing score :  0.15538785394268484
```

In [ ]:
```
#Graphically representing Actual vs Predicted
KGraph = pd.DataFrame({'Predicted':y2_pred, 'Actual': y2_test})
KGraph = KGraph.reset_index()
KGraph = KGraph.drop(['index'], axis=1)

fig= plt.figure(figsize=(6,4))
sns.lineplot(data=KGraph[:120])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa8226710>
```



XGBRegressor

In [ ]:
```
#XGBRegressor
xg_reg = xgb.XGBRegressor(objective ='reg:squarederror', colsample_bytree = 1,eta=0.3, learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 2000)
```

In [ ]:
```
#Fitting & Predicting the Model
xg_reg.fit(X2_train,y2_train)
y2_pred = xg_reg.predict(X2_test)
```

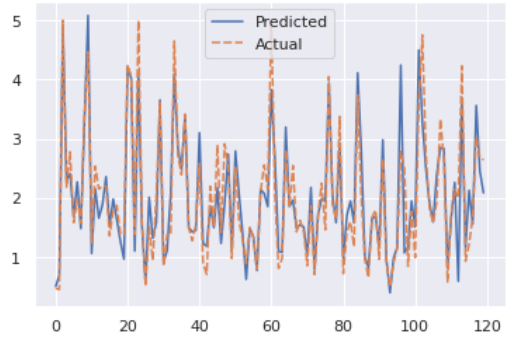
In [ ]:
```
#Producing R2
```

```
r2xgb=r2_score(y2_test,y2_pred)
print('the R squared of the xgboost method is:', r2xgb)
```

```
the R squared of the xgboost method is: 0.8461485016011139
```

In [ ]:
```
#Graphically representing Actual vs Predicted
grp = pd.DataFrame({'Predicted':y2_pred,'Actual':y2_test})
grp = grp.reset_index()
grp = grp.drop(['index'],axis=1)
fig= plt.figure(figsize=(6,4))
sns.lineplot(data=grp[:120])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa752b050>
```



3.5.3 Percentile Based Technique

In [ ]:
```
#Splitting the Data
X2_train, X2_test, y2_train, y2_test = train_test_split(X2_P,y2_P,test_size=0.2,random_state=42)
print("X_train shape {} and size {}".format(X2_train.shape,X2_train.size))
print("X_test shape {} and size {}".format(X2_test.shape,X2_test.size))
print("y_train shape {} and size {}".format(y2_train.shape,y2_train.size))
print("y_test shape {} and size {}".format(y2_test.shape,y2_test.size))
```

```
X_train shape (16512, 8) and size 132096
X_test shape (4128, 8) and size 33024
y_train shape (16512,) and size 16512
y_test shape (4128,) and size 4128
```

In [ ]:
```
def percentile_capping(df, cols, from_low_end, from_high_end):
  for col in cols:
    stats.mstats.winsorize(a=california_housing.frame_P[col], limits=(from_low_end, from_high_end), inplace=True)
```

In [ ]:
```
from scipy import stats
percentile_capping(california_housing.frame_P, X2_train, 0.01, 0.01)
```

Linear Regression

In [ ]:
```
#Linear Model
#Linear Regression
LinRegModel = LinearRegression(n_jobs=-1)
#Fit the model to the training data (learn the coefficients)
LinRegModel.fit(X2_train,y2_train)
#Print the intercept and coefficients
print("Intercept: "+str(LinRegModel.intercept_))
print("Coefficients: "+str(LinRegModel.coef_))
```

```
Intercept: -37.02327770606412
Coefficients: [ 4.48674910e-01  9.72425752e-03 -1.23323343e-01  7.83144907e-01
 -2.02962058e-06 -3.52631849e-03 -4.19792487e-01 -4.33708065e-01]
```

In [ ]:
```
#Prediction on Test Data
y2_pred = LinRegModel.predict(X2_test)

#Predictions and Test Values
print("Length of y2 Predictions:", len(y2_pred))
print("Length of y2 Tests:", len(y2_test))

test = pd.DataFrame({'Predicted':y2_pred,'Actual':y2_test})
test.head()
```

```
Length of y2 Predictions: 4128
Length of y2 Tests: 4128
```

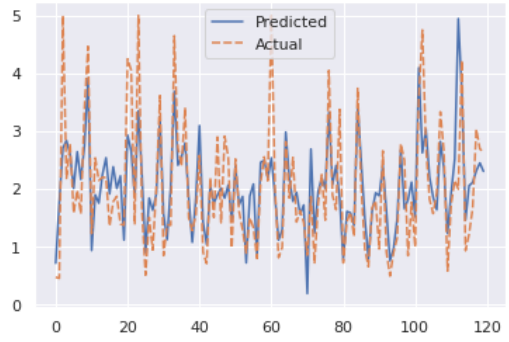
Out[ ]:

	Predicted	Actual
20046	0.719123	0.47700
3024	1.764017	0.45800
15663	2.709659	5.00001
20484	2.838926	2.18600

	Predicted	Actual
9814	2.604657	2.78000

In [ ]:
```
#Charting the Predictions vs Actual
fig= plt.figure(figsize=(6,4))
test = test.reset_index()
test = test.drop(['index'],axis=1)
sns.lineplot(data=test[:120], sort=False)
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8fa75b7a50>



Logstic Regression

In [ ]:
```
#Logistic Regression
Logreg2 = LogisticRegression(solver='lbfgs', max_iter=10000)
Logreg2.fit(X2_train, np.ceil(y2_train))
```

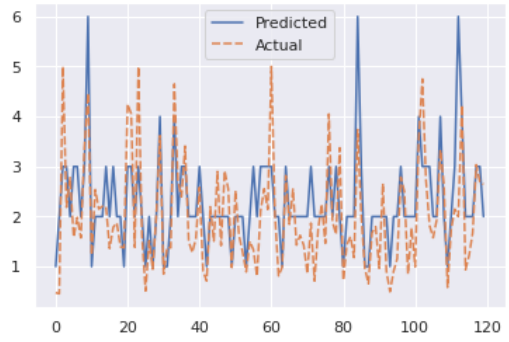
Out[ ]: LogisticRegression(max_iter=10000)

In [ ]:
```
y2_pred = Logreg2.predict(X2_test)
print('Accuracy: %.3f' % accuracy_score(np.ceil(y2_test), y2_pred))
print('Precision: %.3f' % precision_score(np.ceil(y2_test), y2_pred, average="weighted"))
print('Recall: %.3f' % recall_score(np.ceil(y2_test), y2_pred, average="weighted"))
print('F-measure: %.3f' % f1_score(np.ceil(y2_test), y2_pred, average="weighted"))
```

```
Accuracy: 0.568
Precision: 0.535
Recall: 0.568
F-measure: 0.539
```

In [ ]:
```
#Graphically representing Actual vs Predicted
LogGraph = pd.DataFrame({'Predicted':y2_pred, 'Actual': y2_test})
LogGraph = LogGraph.reset_index()
LogGraph = LogGraph.drop(['index'], axis=1)

fig= plt.figure(figsize=(6,4))
sns.lineplot(data=LogGraph[:120])
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8fa7155190>



Kneighbours

In [ ]:
```
#KNeighbors Regression
knn_regressor = KNeighborsRegressor(n_neighbors = 10).fit(X2_train, y2_train)
```

In [ ]:
```
print("Training score : ", knn_regressor.score(X2_train, y2_train))
```

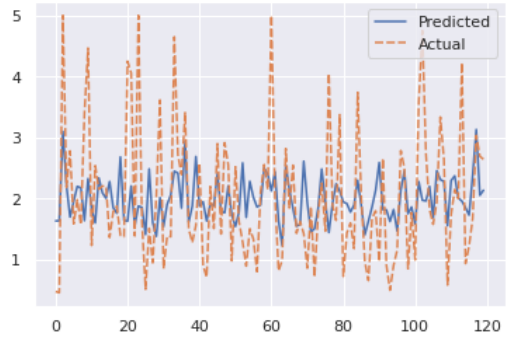
```
Training score :  0.3216293223790314
```

In [ ]:
```
y2_pred = knn_regressor.predict(X2_test)
print("testing score : ", r2_score(y2_test, y2_pred))
```

```
testing score :  0.15538785394268484
```

In [ ]:
```
#Graphically representing Actual vs Predicted
KGraph = pd.DataFrame({'Predicted':y2_pred, 'Actual': y2_test})
KGraph = KGraph.reset_index()
KGraph = KGraph.drop(['index'], axis=1)

fig= plt.figure(figsize=(6,4))
sns.lineplot(data=KGraph[:120])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa7b1fb50>
```



XGBRegressor

In [ ]:
```
#XGBRegressor
xg_reg = xgb.XGBRegressor(objective ='reg:squarederror', colsample_bytree = 1,eta=0.3, learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 2000)
```

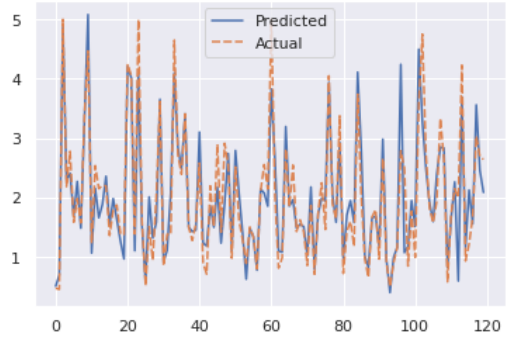
In [ ]:
```
#Fitting & Predicting the Model
xg_reg.fit(X2_train,y2_train)
y2_pred = xg_reg.predict(X2_test)
```

In [ ]:
```
#Producing R2
r2xgb=r2_score(y2_test,y2_pred)
print('the R squared of the xgboost method is:', r2xgb)
```

```
the R squared of the xgboost method is: 0.8461485016011139
```

In [ ]:
```
#Graphically representing Actual vs Predicted
grp = pd.DataFrame({'Predicted':y2_pred,'Actual':y2_test})
grp = grp.reset_index()
grp = grp.drop(['index'],axis=1)
fig= plt.figure(figsize=(6,4))
sns.lineplot(data=grp[:120])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa747f090>
```



3.5.4 Windzorization

In [ ]:
```
#Splitting the Data
X2_train, X2_test, y2_train, y2_test = train_test_split(X2_W,y2_W,test_size=0.2,random_state=42)
print("X_train shape {} and size {}".format(X2_train.shape,X2_train.size))
print("X_test shape {} and size {}".format(X2_test.shape,X2_test.size))
print("y_train shape {} and size {}".format(y2_train.shape,y2_train.size))
print("y_test shape {} and size {}".format(y2_test.shape,y2_test.size))
```

```
X_train shape (16512, 8) and size 132096
X_test shape (4128, 8) and size 33024
y_train shape (16512,) and size 16512
y_test shape (4128,) and size 4128
```

In [ ]:
```
#Explore different quantiles at the upper end
print('90% quantile:   ', california_housing.frame.quantile(0.90))       #6.15
```

```
print('92.5% quantile: ', california_housing.frame.quantile(0.925))      #6.6
print('95% quantile:   ', california_housing.frame.quantile(0.95))       #7.3
print('97.5% quantile: ', california_housing.frame.quantile(0.975))      #8.47
print('99% quantile:   ', california_housing.frame.quantile(0.99))       #10.59
print('99.9% quantile: ', california_housing.frame.quantile(0.999))      #15.00
```

```
90% quantile:    MedInc           6.159210
HouseAge         46.000000
AveRooms          6.961188
AveBedrms         1.172727
Population     2566.000000
AveOccup          3.885273
Latitude         38.480000
Longitude      -117.250000
MedHouseVal       3.766000
Name: 0.9, dtype: float64
92.5% quantile:  MedInc           6.600437
HouseAge         49.000000
AveRooms          7.235487
AveBedrms         1.205882
Population     2855.000000
AveOccup          4.094140
Latitude         38.630000
Longitude      -117.150000
MedHouseVal       4.192000
Name: 0.925, dtype: float64
95% quantile:    MedInc           7.300305
HouseAge         52.000000
AveRooms          7.640247
AveBedrms         1.273006
Population     3288.000000
AveOccup          4.333417
Latitude         38.960000
Longitude      -117.080000
MedHouseVal       4.898100
Name: 0.95, dtype: float64
97.5% quantile:  MedInc           8.470930
HouseAge         52.000000
AveRooms          8.344463
AveBedrms         1.487222
Population     4197.025000
AveOccup          4.725650
Latitude         39.740000
Longitude      -116.950000
MedHouseVal       5.000010
Name: 0.975, dtype: float64
99% quantile:    MedInc          10.596540
HouseAge         52.000000
AveRooms         10.357033
AveBedrms         2.127541
Population     5805.830000
AveOccup          5.394812
Latitude         40.626100
Longitude      -116.290000
MedHouseVal       5.000010
Name: 0.99, dtype: float64
99.9% quantile:  MedInc           15.000100
HouseAge         52.000000
AveRooms         34.199698
AveBedrms         6.617232
Population     10372.681000
AveOccup          13.630443
Latitude          41.753610
Longitude       -114.646390
MedHouseVal       5.000010
Name: 0.999, dtype: float64
```

In [ ]:
```
features = ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup',]
features
```

Out[ ]:
```
['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup']
```

In [ ]:
```
def percentile_capping(df, cols, from_low_end, from_high_end):

    for col in cols:

       stats.mstats.winsorize(a=df[col], limits=(from_low_end, from_high_end), inplace=True)
```

In [ ]:
```
percentile_capping(california_housing.frame, features, 0.1, 0.1)
```

Linear Regression

In [ ]:
```
#Linear Regression
LinRegModel = LinearRegression(n_jobs=-1)
#Fit the model to the training data (learn the coefficients)
LinRegModel.fit(X2_train,y2_train)
#Print the intercept and coefficients
print("Intercept: "+str(LinRegModel.intercept_))
print("Coefficients: "+str(LinRegModel.coef_))
```

```
Intercept: -37.02327770606412
Coefficients: [ 4.48674910e-01  9.72425752e-03 -1.23323343e-01  7.83144907e-01
```

```
-2.02962058e-06 -3.52631849e-03 -4.19792487e-01 -4.33708065e-01]
```

In [ ]:
```
#Prediction on Test Data
y2_pred = LinRegModel.predict(X2_test)

#Predictions and Test Values
print("Length of y2 Predictions:", len(y2_pred))
print("Length of y2 Tests:", len(y2_test))

test = pd.DataFrame({'Predicted':y2_pred,'Actual':y2_test})
test.head()
```

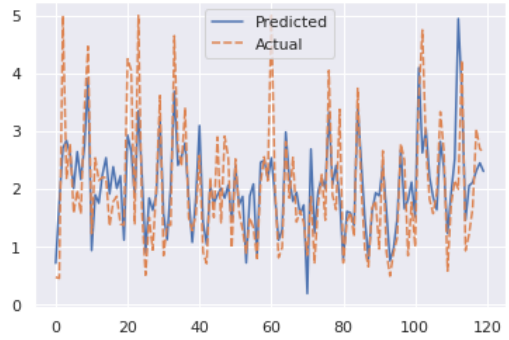
```
Length of y2 Predictions: 4128
Length of y2 Tests: 4128
```

Out[ ]:

	Predicted	Actual
20046	0.719123	0.47700
3024	1.764017	0.45800
15663	2.709659	5.00001
20484	2.838926	2.18600
9814	2.604657	2.78000

In [ ]:
```
#Charting the Predictions vs Actual
fig= plt.figure(figsize=(6,4))
test = test.reset_index()
test = test.drop(['index'],axis=1)
sns.lineplot(data=test[:120], sort=False)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa7ab0190>
```



Logstic Regression

In [ ]:
```
#Logistic Regression
Logreg2 = LogisticRegression(solver='lbfgs', max_iter=10000)
Logreg2.fit(X2_train, np.ceil(y2_train))
```

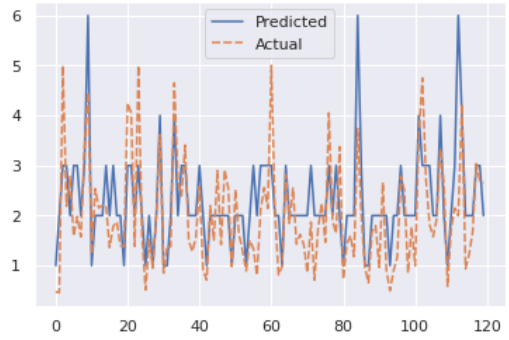
Out[ ]:
```
LogisticRegression(max_iter=10000)
```

In [ ]:
```
y2_pred = Logreg2.predict(X2_test)
print('Accuracy: %.3f' % accuracy_score(np.ceil(y2_test), y2_pred))
print('Precision: %.3f' % precision_score(np.ceil(y2_test), y2_pred, average="weighted"))
print('Recall: %.3f' % recall_score(np.ceil(y2_test), y2_pred, average="weighted"))
print('F-measure: %.3f' % f1_score(np.ceil(y2_test), y2_pred, average="weighted"))
```

```
Accuracy: 0.568
Precision: 0.535
Recall: 0.568
F-measure: 0.539
```

In [ ]:
```
#Graphically representing Actual vs Predicted
LogGraph = pd.DataFrame({'Predicted':y2_pred, 'Actual': y2_test})
LogGraph = LogGraph.reset_index()
LogGraph = LogGraph.drop(['index'], axis=1)

fig= plt.figure(figsize=(6,4))
sns.lineplot(data=LogGraph[:120])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa7ca66d0>
```



Kneighbours

In [ ]:
```
#KNeighbors Regression
knn_regressor = KNeighborsRegressor(n_neighbors = 10).fit(X2_train, y2_train)
```

In [ ]:
```
print("Training score : ", knn_regressor.score(X2_train, y2_train))
```

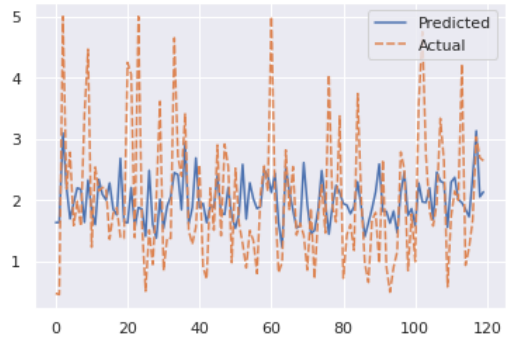
```
Training score :  0.3216293223790314
```

In [ ]:
```
y2_pred = knn_regressor.predict(X2_test)
print("testing score : ", r2_score(y2_test, y2_pred))
```

```
testing score :  0.15538785394268484
```

In [ ]:
```
#Graphically representing Actual vs Predicted
KGraph = pd.DataFrame({'Predicted':y2_pred, 'Actual': y2_test})
KGraph = KGraph.reset_index()
KGraph = KGraph.drop(['index'], axis=1)

fig= plt.figure(figsize=(6,4))
sns.lineplot(data=KGraph[:120])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa76a9d90>
```



XGBRegressor

In [ ]:
```
#XGBRegressor
xg_reg = xgb.XGBRegressor(objective ='reg:squarederror', colsample_bytree = 1,eta=0.3, learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 2000)
```

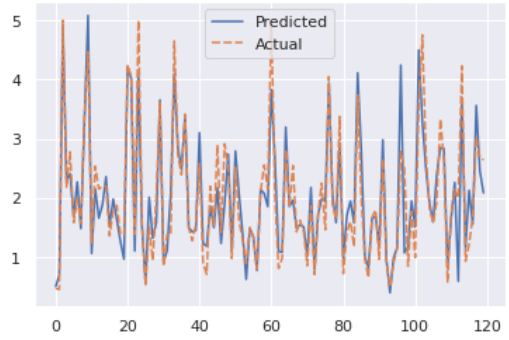
In [ ]:
```
#Fitting & Predicting the Model
xg_reg.fit(X2_train,y2_train)
y2_pred = xg_reg.predict(X2_test)
```

In [ ]:
```
r2xgb=r2_score(y2_test,y2_pred)
print('the R squared of the xgboost method is:', r2xgb)
```

```
the R squared of the xgboost method is: 0.8461485016011139
```

In [ ]:
```
#Graphically representing Actual vs Predicted
grp = pd.DataFrame({'Predicted':y2_pred,'Actual':y2_test})
grp = grp.reset_index()
grp = grp.drop(['index'],axis=1)
fig= plt.figure(figsize=(6,4))
sns.lineplot(data=grp[:120])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa7b54150>
```



4. Summary

4.1 Theory

Before regression analysis could be applied to the datasets, they first had to be preprocessed.

This involved detecting NANs, dealing with imbalances in the data and applying oulier techniques. Having previously detected NANs and dealt with imbalances, the outlier techniques were a new concept.

The first technique used was the Z-score based technique. This was easy to understand as we had came across Z-scores before. This technique set an upper and lower bound for the Z-score of a data point, and if the Z-score was outside these limits, it was determined an outlier and removed. The second technique used was the IQR based filtering. This technique also set upper and lower limits, but uses the inter-quartile range rather than the Z-score. One benefit of this technique is its use in image denoising. The third outlier detection technique used was the percentile-based technique. This technique is similar to the two mentioned above but sets limits directly. For example, identifying all data points that fall below the bottom x% e.g., 5% and above the top x_2% e.g., 95%. The final technique called winsorisation is similar to percentile-based as it uses percentiles, the kth percentile is set but rather than removing values that lie below and above this, these values are set to the value that lies on the kth percentile. This technique can be useful when comparing winsorized means and other stats to the original mean and other stats as it can identify variables that contains contaminated data (Wicklin, 2017).

Regression is a statistical method that attempts to determine the strength of a relationship between one dependent variavle Y and one or more independent variables x_i.

Four different methods of regression analysis were applied to two datasets in this report. The four methods used were linear, logistic, XGBoost and K-nearest neighbours. The first 3 methods had been previously been applied to classification problems so were familiar to the team. However, the K-nearest neighbours algorithm was new to us. This algorithm is widely used in a number of industries, such as crop prediction in agriculture, facial regonition in technology and diagnosing disases in medicine.

As a regression technique, its measures the distance between a new point and the k other closest points and determines the output by getting the average of the k these points. The benefits of this technique are that it is quick, simple to interpret, versatile and highly accurate. Some of the disadvantages of this algorithm are that the accuracy depends on quality data and it can be sensitive to the scale of the data (Chatterjee, 2020).

4.2 Pros & Cons of Datasets

One con of the California Housing Dataset is that the dependent variable is continuous. This causes problems to arise when applying the Logistic Regression function to the datasets. The columns of Longitude and Latitude, while particularly useful gor graphing the locations, aren't very helpful when running models such as Linear Regression.

4.3 Findings & Conclusions

Synthetic dataset

When using the mean squared error approach of predicting the accuracy of each of the regression, we want a low value for the mean squared error. The lower the mean squared error, the higher the accuracy of the prediction.

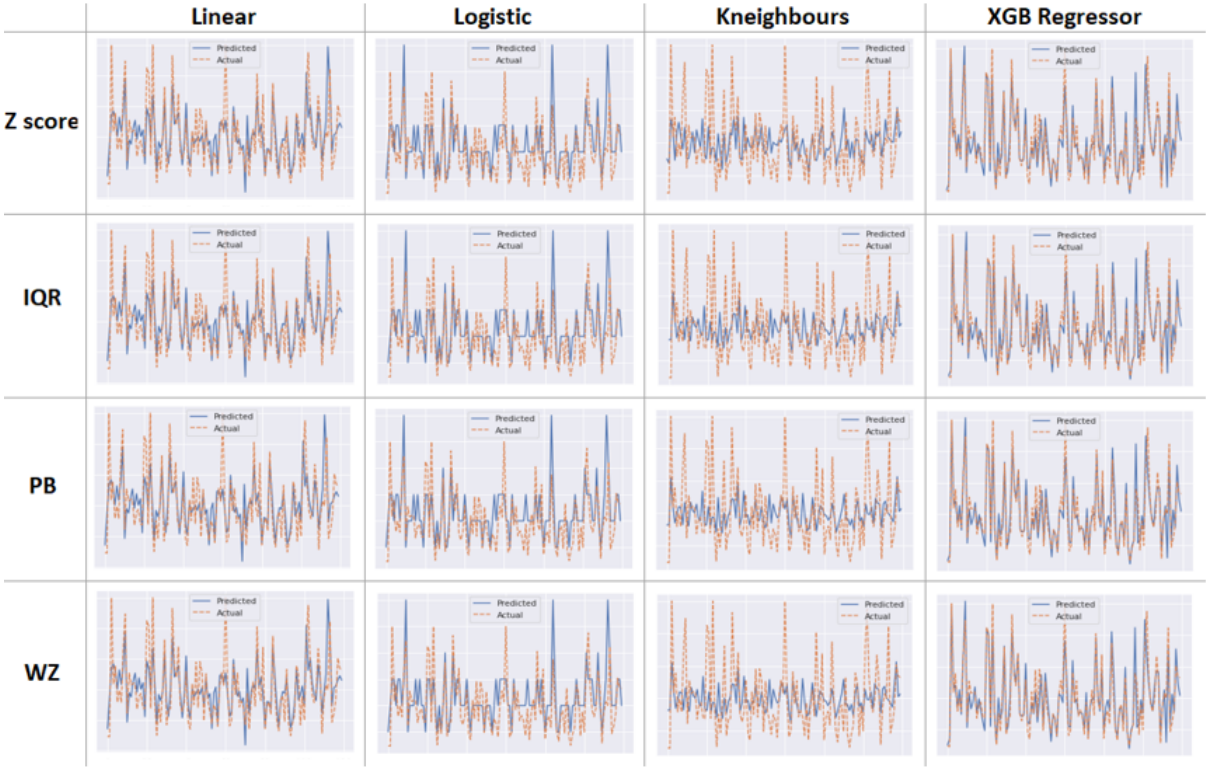
Thus, we can conclude that the most accurate regression model was the XGBoost Regressor when the dataset still had the outliers contained in it. After the outliers had been removed through the IQR based filtering method and each of the regression models were ran again, the Logistic Regression was the most accurate as it had the lowest score for the mean squared error.

Overall, the regression models of Linear Regression, XGB as a Regressor, KNeighbours as a Regressor, and Logistic Regression appeared to work better after the outliers had been removed from the dataset. We can come to the conclusion that removing outliers makes the data more accurate. While outliers can be informative, the increase the variability of data, which decrease the statistical power. Therefore, the removal of outliers can make our results more statistically significant.

California Housing dataset

The target variable is the median house value for California districts.

This dataset was derived from the 1990 U.S. census, using one row per census block group. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people).



The above chart displays the Actual vs Predictions for each Model, under each outlier technique. What's quite interesting to note, is that the models perform quite similar under the various outlier techniques. This would indicate, that all 4 of them did a satisfactory job in dealing with the outliers.

In terms of the models themselves, it's clear that the award winning XGB Regressor performed the best. It recorded scores of around 84-86% under each technique. On the contrary, Logtistic Regression was clearly the worst, with scores of .321 for training, and .155 for testing. While these scores are particularly poor, it''s also understandable, as we had to round each dependent variable using np.floor, making the results highly inaccurate.

4.4 Hints & Tips

The use of np.ceil, np.floor or np.round were particularly useful when applying the Logistic Regression to both datasets. As the y varaible was continuous in both, without using such functions, an error would arise.

Furthermore, plotting the actual vs. predicted for each technique and method allowed us to visually represent how accurate or inaccurate they were.

Bibliography

Albon, C. (2018) Machine Learning with Python cookbook: practical solutions from preprocessing to deep learning. First edition. Sebastopol, CA: O'Reilly Media. [accessed 21 Oct 2021].

Bonthu, H., 2021. Detecting and Treating Outliers | Treating the odd one out!. [Blog] Analytics Vidhya, Available at: https://www.analyticsvidhya.com/blog/2021/05/detecting-and-treating-outliers-treating-the-odd-one-out/ [Accessed 27 October 2021].

Chatterjee, M. (2020). The Introduction of KNN Algorithm | What is KNN Algorithm?. GreatLearning Blog: Free Resources what Matters to shape your Career!. Retrieved 30 October 2021, from https://www.mygreatlearning.com/blog/knn-algorithm-introduction/.

Hoppen, J., 2021. What are outliers and how to treat them in Data Analytics. [Blog] Aquarela, Available at: https://www.aquare.la/en/what-are-outliers-and-how-to-treat-them-in-data-analytics/ [Accessed 27 October 2021].

Horsch, A., 2021. Detecting and Treating Outliers In Python — Part 3. [Blog] Towards Data Science, Available at: https://towardsdatascience.com/detecting-and-treating-outliers-in-python-part-3-dcb54abaf7b0 [Accessed 27 October 2021].

Linear Regression (n.d.) ibm.com, available: https://www.ibm.com/topics/linear-regression [accessed 22 Oct 2021].

Maini, E. (2020) Z score for Outlier Detection, available: https://www.geeksforgeeks.org/z-score-for-outlier-detection-python/ [accessed 20 Oct 2021].

Muller, A.C., and Guid, S. (2016) Introduction to Machine Learning with Python: A Guide For Data Scientists, First edition., Sebastopol, CA: O'Reilly Media. [accessed 08 Oct 2021].

Păpăluță, V., 2020. What's the best way to handle NaN values?. [Blog] Towards Data Science, Available at: https://towardsdatascience.com/whats-the-best-way-to-handle-nan-values-62d50f738fc [Accessed 27 October 2021].

Shukla, S. (2021) Regression and Classification Supervised Machine Learning, available: https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/ [accessed 20 Oct 2021].

Singh, A. (2021). K-Nearest Neighbors Algorithm | KNN Regression Python. Analytics Vidhya. Retrieved 30 October 2021, from https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/.

Warudkar, H. (2020) Dealing With Outliers In Machine Learning, available: https://expressanalytics.com/blog/outliers-machine-learning/ [accessed 20 Oct 2021].

What is Winsorization (2016) statisticshowto.com, available: https://www.statisticshowto.com/winsorize/ [accessed 22 Oct 2021].

Wicklin, R. (2017). Winsorization: The good, the bad, and the ugly. The DO Loop. Retrieved 31 October 2021, from https://blogs.sas.com/content/iml/2017/02/08/winsorization-good-bad-and-ugly.html.