UCLA

ECE 219 - Large-Scale Data Mining: Models and Algorithms
Winter 2021

Project 4: Regression Analysis

Authors:
Swapnil Sayan Saha (UID: 605353215)
Grant Young (UID: 505627579)

Question 1:

In this question, we are asked to plot the heatmaps of Pearson correlation matrix of dataset columns for bike-sharing, suicide rates and video transcoding datasets:

- The bike-sharing dataset contains two-year (2011 and 2012) daily (and hourly) bike rental counts from a Washington D.C.-based bike rental company along with environmental and seasonal information. We use the daily bike rental counts in this project, which has 731 samples arranged in tabular fashion. The target columns include:
    - **casual**: numerical, number of casual users.
    - **registered**: numerical, number of registered users.
    - **cnt**: numerical, total number of bikes rented including casual and registered users. This is the target variable we use later during training regression models (section 3.1.1).

  The 11 feature columns (excluding record index (**instant**) and date (**dteday**), both of which have the highest cardinalities and do not contribute any semantic information for regression) include:
    - **season**: categorical, 1. Spring 2. Summer 3. Fall 4. Winter.
    - **yr**: categorical, 0. 2011, 1. 2012.
    - **mnth**: categorical, 1 to 12 indicates months January through December.
    - **holiday**: categorical, indicates whether the day is holiday or not (1 or 0).
    - **weekday**: categorical, day of the week (0 to 6).
    - **workingday**: categorical, takes value of 1 if a day is neither weekend nor holiday is 1, otherwise takes value of 0.
    - **weathersit**: categorical, takes value from 1 to 4, weather becomes more extreme as value increases.
        - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
        - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
        - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
        - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
    - **temp** : numerical, normalized temperature in Celsius.
    - **atemp**: numerical, normalized feeling temperature in Celsius.
    - **hum**: numerical, normalized humidity.
    - **windspeed**: numerical, normalized wind speed.
- The suicides rates overview (1985 to 2016) dataset presents the country-wise and time-wise count of the total number of suicides and suicides per 100,000 population across different cohorts in the global socioeconomic spectrum. The dataset contains 27820 observations. The target variables include:
    - **suicides_no**: numerical, total number of suicides in a country for a specific year, age-group, sex and generation.

- **suicides/100k pop**: numerical, suicides_no divided by the population within that country for a specific year, age-group, sex and generation. This is the target variable we use later during training regression models (section 3.1.1).

The 8 feature columns include:

- **country**: categorical, specified as string (with high cardinality), 101 unique countries.
- **year**: numerical, year from 1987 to 2016.
- **sex**: categorical, male or female, specified as string.
- **age**: categorical, specifies 6 different age groups, namely 5-14, 15-24, 25-34, 35-54, 55-74 and 75+ years.
- **generation**: categorical, specifies 6 different generations, namely Generation X, Silent, Millenials, Boomers, G.I. Generation and Generation Z.
- **population**: numerical, total population for a specific country, year, age-group, sex and generation.
- **gdp_for_year**: numerical, gross domestic product in US dollars for a specific country in a specific year.
- **gdp_per_capita**: numerical, gross domestic product in US dollars per capita for a specific country, year, age-group, sex and generation.

- The video transcoding dataset contains input and output video characteristics along with their transcoding time and memory resource requirements for 68784 samples arranged in tabular fashion in the "transcoding_mesurment.tsv" file. The target variables include:
  - **umem**: numerical, total codec allocated memory for transcoding.
  - **utime**: numerical, total transcoding time for transcoding. This is the target variable we use later during training regression models (section 3.1.1).

  The 19 feature columns include:
  - **duration**: numerical, duration of video in second.
  - **codec**: categorical, coding standard used for the input video: flv, h264, mpeg4, vp8.
  - **height**: numerical, input height of video in pixels.
  - **width**: numerical, input width of video in pixels.
  - **bitrate**: numerical, bits that are conveyed or processed per unit of time for input video.
  - **frame_rate**: numerical, input video frame rate (fps).
  - **i**: numerical, number of i frames in the video, where i frames are the least compressible but don't require other video frames to decode.
  - **p**: numerical, number of p frames in the video, where p frames can use data from previous frames to decompress and are more compressible than i frames.
  - **b**: numerical, number of b frames in the video, where b frames can use both previous and forward frames for data reference to get the highest amount of data compression.
  - **frames**: numerical, total number of frames in video.
  - **i-size**: numerical, total size in bytes of i frames.
  - **p-size**: numerical, total size in bytes of p frames.

- **b-size**: numerical, total size in bytes of b frames.
- **size**: numerical, total size of video in bytes.
- **o-codec**: categorical, output codec used for transcoding, flv, h264, mpeg4, vp8
- **o-bitrate**: numerical, output bitrate used for transcoding.
- **o-framerate**: numerical, output framerate used for transcoding.
- **o-width**: numerical, output width in pixel used for transcoding.
- **o-height**: numerical, output height in pixel used for transcoding.

The Pearson correlation coefficient is a measure of linear correlation among two variables, ranging between -1 and 1. -1 indicates a perfect negative correlation (as one feature decreases, the other increases linearly), 0 indicates no correlation and 1 indicates a perfect position correlation (both features increase or decrease linearly with each other).

$$\rho_{X,Y} = \frac{\mathbb{E}[(X - \mu_X)(Y - \mu_Y)}{\sigma_X \sigma_Y}$$

where X and Y are the concerned features, μ is the mean of the feature, σ is the standard deviation and $\mathbb{E}$ is the expected value.

Figure 1 shows the heatmap of Pearson correlation matrix of dataset columns for the bike-sharing dataset. To plot the heatmap, we used `pandas-profiling`, which automatically generated the plot for valid features. The profile was saved as an HTML file for permanent access.

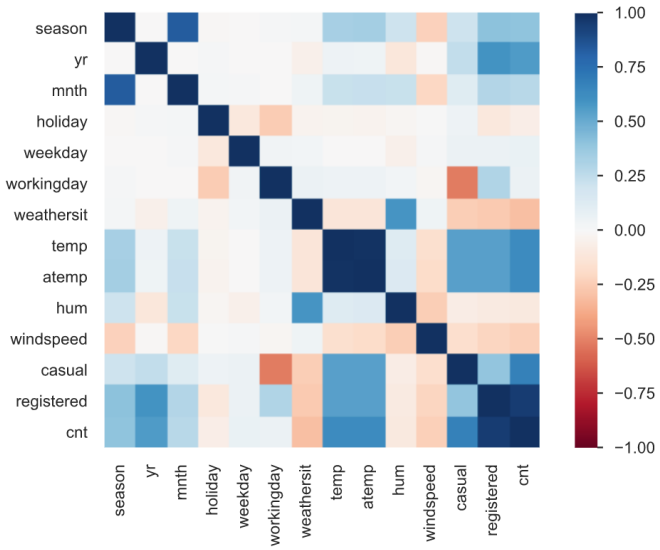


Figure 1: Pearson correlation heatmap for the bike-sharing dataset (generated by `pandas-profiling`).

For each target variable, the features with the highest absolute correlation are as follows:

- **casual**: **temp**, **atemp** and **workingday** have the highest absolute correlations (along with registered and cnt, but these two are target variables). The intuition is that people prefer leisurely biking when the

weather is comfortable and warm in terms of both absolute temperatures and "real-feel" temperatures, while taking public transportation or cars to go to work in urban areas such as Washington D.C., where the data is from.

- **registered: temp, atemp and yr** have the highest absolute correlations (along with casual and cnt, but these two are target variables). The intuition behind 'temp' and 'atemp' holds from previous question, while the reason for more bikes being rented in 2012 than 2011 is probably due to the occurrence of Hurricane Sandy in Washington D.C. in 2012, which may have caused a shortage of other forms of transportation and hence an increased number of registrations in the bike company.
- **cnt: temp, atemp and yr** have the highest absolute correlations (along with casual and registered, but these two are target variables). Since the total number of bike rentals is directly dependent on the number of casual and registered users, the same intuition explained in case of 'registered' and 'casual' for 'temp', 'atemp' and 'yr' holds true for 'cnt' as well.

Figure 2 shows the heatmap of Pearson correlation matrix of dataset columns for the suicide-rates dataset. Note that `pandas-profiling` automatically omits the categorical variables country, year, sex, age and generation as they were specified as string values, which `pandas-profiling` cannot process to generate heatmaps.

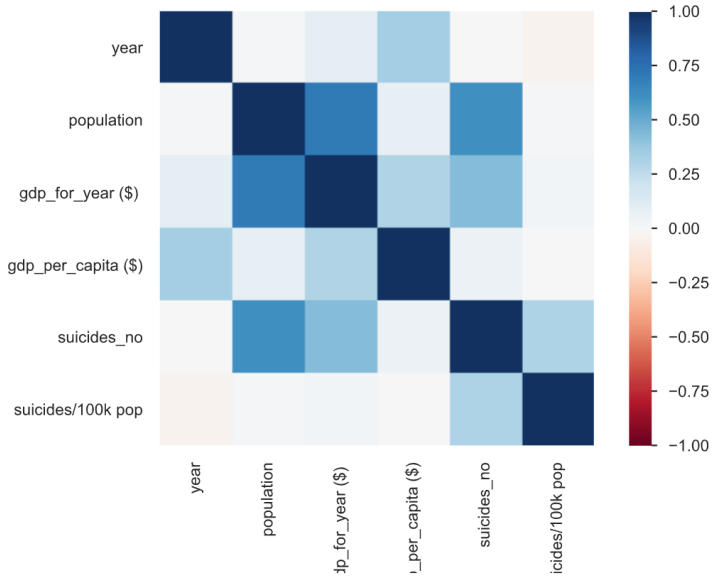


Figure 2: Pearson correlation heatmap for the suicide-rates dataset (generated by `pandas-profiling`).

For each target variable, the features with the highest absolute correlation are as follows:

- **suicides_no**: **population and gdp_for_year** have the highest absolute correlation. Intuitively, a larger population indicates a greater count for the absolute number of suicides (which is why it is recommended to use suicides/100k population, which is normalized on the population count). In addition, while a larger GDP indicates higher economic prosperity, the happiness index may not

correlate with that of the development metrics. In fact, higher GDP indicates more productive work hours for the population, which can reduce the amount of time a particular person can enjoy for self-care and mental health checks. Higher number of work hours without leisure, breaks or self-care can lead to depression and suicidal thoughts.

- **suicides/100k population:** We see that suicides_no is the only variable that is correlated with suicides/100k population. This is misleading because `pandas-profiling` has ignored some critical categorical features such as age-group, sex and generation while generating the Pearson correlation heatmap. In fact, we can look at the full picture from ϕ_k correlation heatmap, which is also generated by `pandas-profiling` and works between categorical, ordinal and interval variables:

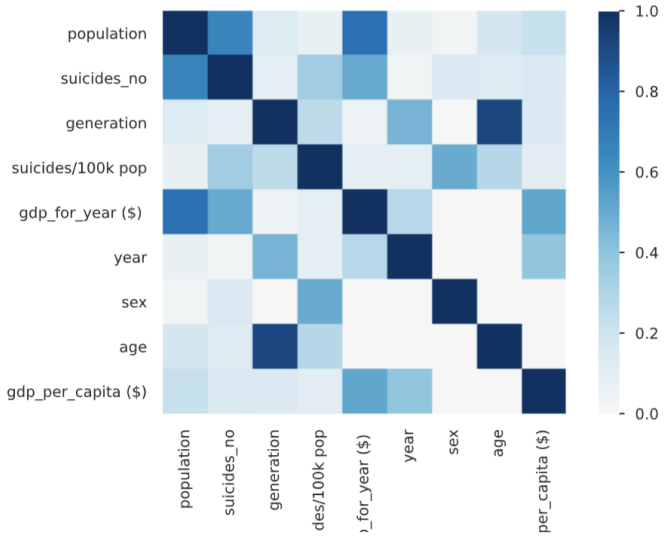


Figure 3: ϕ_k correlation heatmap for the suicide-rates dataset (generated by `pandas-profiling`).

From the ϕ_k correlation heatmap, we can see that for **suicides/100k pop**, we see that **sex** has the highest absolute correlation, followed by **age** and **generation**. Globally, men tend to commit more suicides than women (this is also verified later in Question 5), thus sex has a high correlation with normalized suicide rates. In addition, older people tend to commit more suicides than younger population.

Figure 4 shows the heatmap of Pearson correlation matrix of dataset columns for the video-transcoding dataset. Since `pandas-profiling` omitted 'codec' and 'o-codec' in lieu of these two features being categorical variables, we also show the ϕ_k correlation heatmap for the same dataset. For each target variable, the features with the highest absolute correlation are as follows:

- **umem:** From both the heatmaps, we see that **o_width, o_height** and **o_codec** have the highest absolute correlation (along with utime, but that is a target variable), which is. From intuition, larger output video sizes tend to consume more transcoding memory, while the output codec determines the

compression and decompression memory required for processing the video, with codecs such as mpeg4 providing higher compression rates than h264.

- **utime:** From both the heatmaps, we see that we see that **o_width**, **o_height** and **o_codec** have the highest absolute correlation (along with umem, but that is a target variable). This is expected because larger videos require more transcoding time than smaller videos.

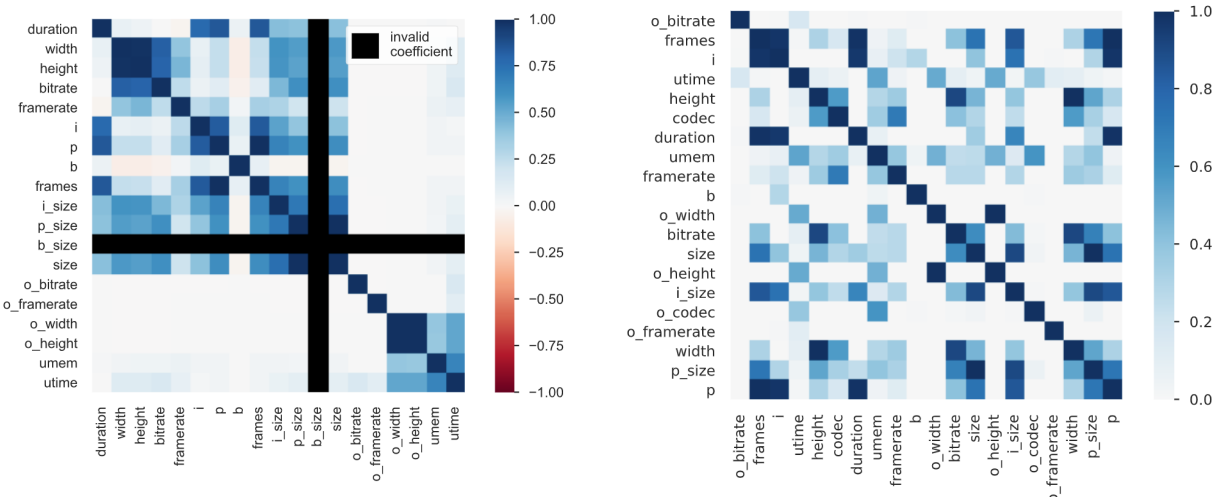


Figure 4: (Left) Pearson correlation heatmap for the video-transcoding dataset, (Right) ϕ_k correlation heatmap for the same dataset (generated by `pandas-profiling`).

Question 2:

In this question, we are asked to plot the histogram of numerical features for all three datasets. We exploit `pandas-profiling` to automatically generate the appropriate histograms for us. The valid numerical features for each dataset include:

- **Bike-sharing dataset**: **temp**, **atemp**, **hum** and **windspeed**, along with **casual**, **registered** and **cnt** (which are target variables).
- **Suicide-rates dataset**: **year**, **population**, **gdp_for_year** and **gdp_per_capita**, along with **suicides_no** and **suicides/100k pop** (which are target variables).
- **Video-transcoding dataset**: **duration, height, width, bitrate, frame_rate, i, p, b, frames, i-size, p-size, b-size, size, o-bitrate, o-framerate, o-width** and **o-height**, along with **umem** and **utime** (which are target variables).

Figure 5 shows the histogram plots of the numerical features for bike-sharing dataset, while Figure 6 shows the histogram plots of the target variables for the same dataset.

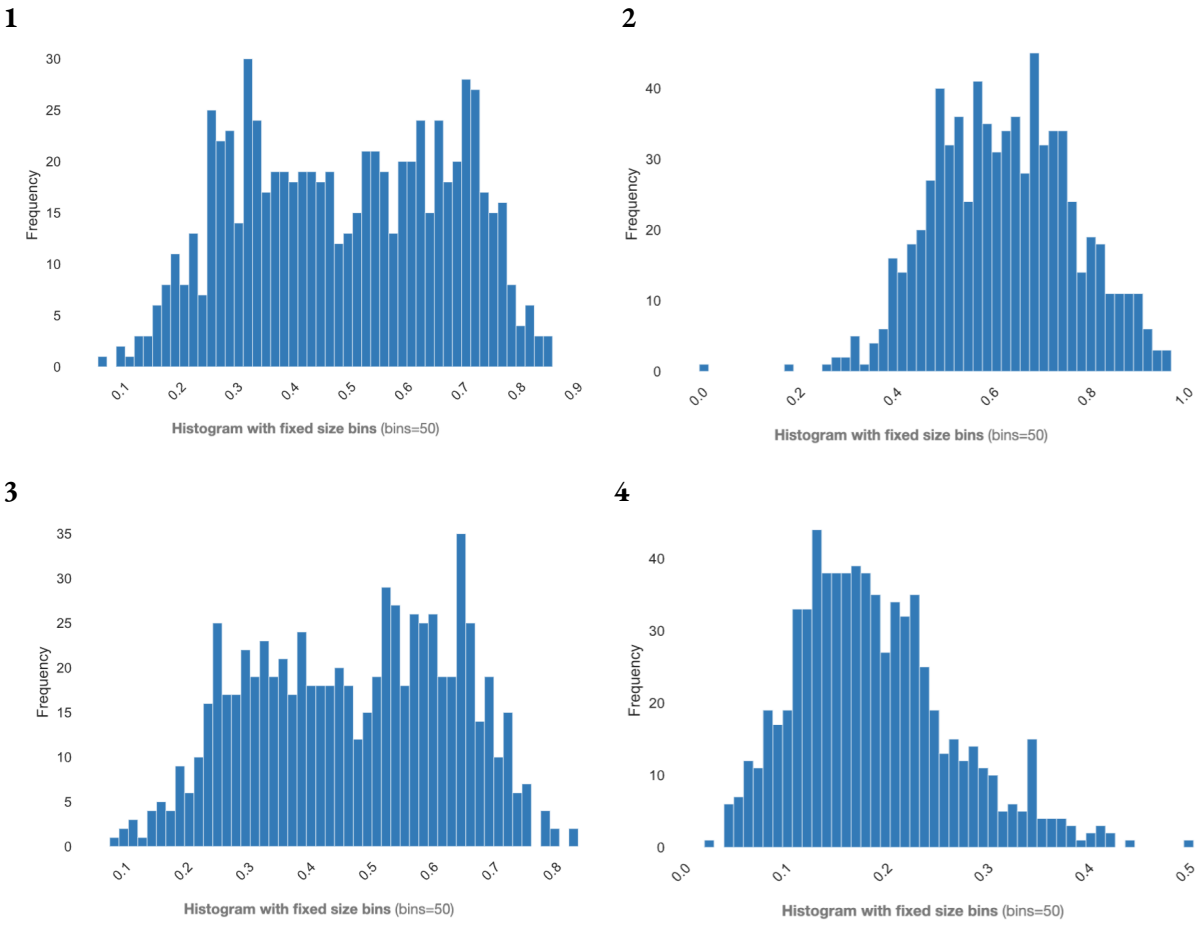


Figure 5: Histogram plots for (1) temp, (2) hum, (3) atemp and (4) windspeed for bike-sharing dataset

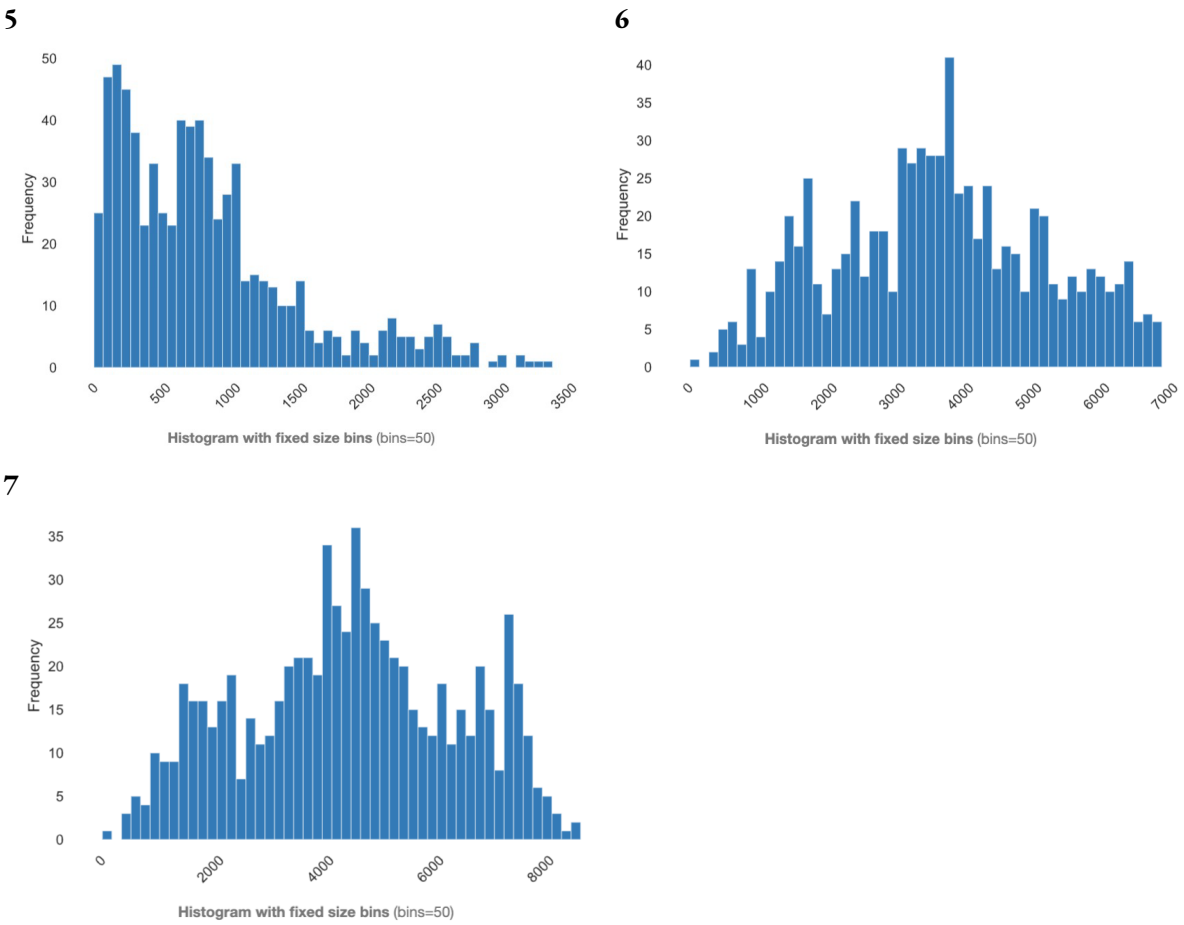


Figure 6: Histogram plots for (5) casual, (6) registered and (7) cnt for bike-sharing dataset.

From Figure 5, we see that 'hum' and 'windspeed' are negatively/left (median > mean) and positively/right (median < mean) skewed respectively, while 'temp' and 'atemp' are bimodally normally distributed without any skewness. Similarly, from Figure 6, in case of target variables, 'registered' and 'cnt' are normally distributed without any skewness, while 'casual' is positively skewed.

Figure 7 shows the histogram plots of the numerical features for suicide-rates dataset, while Figure 8 shows the histogram plots of the target variables for the same dataset. From Figure 7, we see that 'gdp_for_year', 'gdp_per_capita' and population are positively skewed while 'year' is negatively skewed. From Figure 8, in case of target variables, both 'suicides_no' and ' suicides/100k pop' are positively skewed.

Figure 9 shows the histogram plots of the numerical features for video-transcoding dataset, while Figure 10 shows the histogram plots of the target variables for the same dataset. From Figure 9, we can see that most of the features are positively/right skewed. In case of target variables, from Figure 10, we observe that 'utime' is positively skewed, while 'umem' has no skewness and is normally distributed.

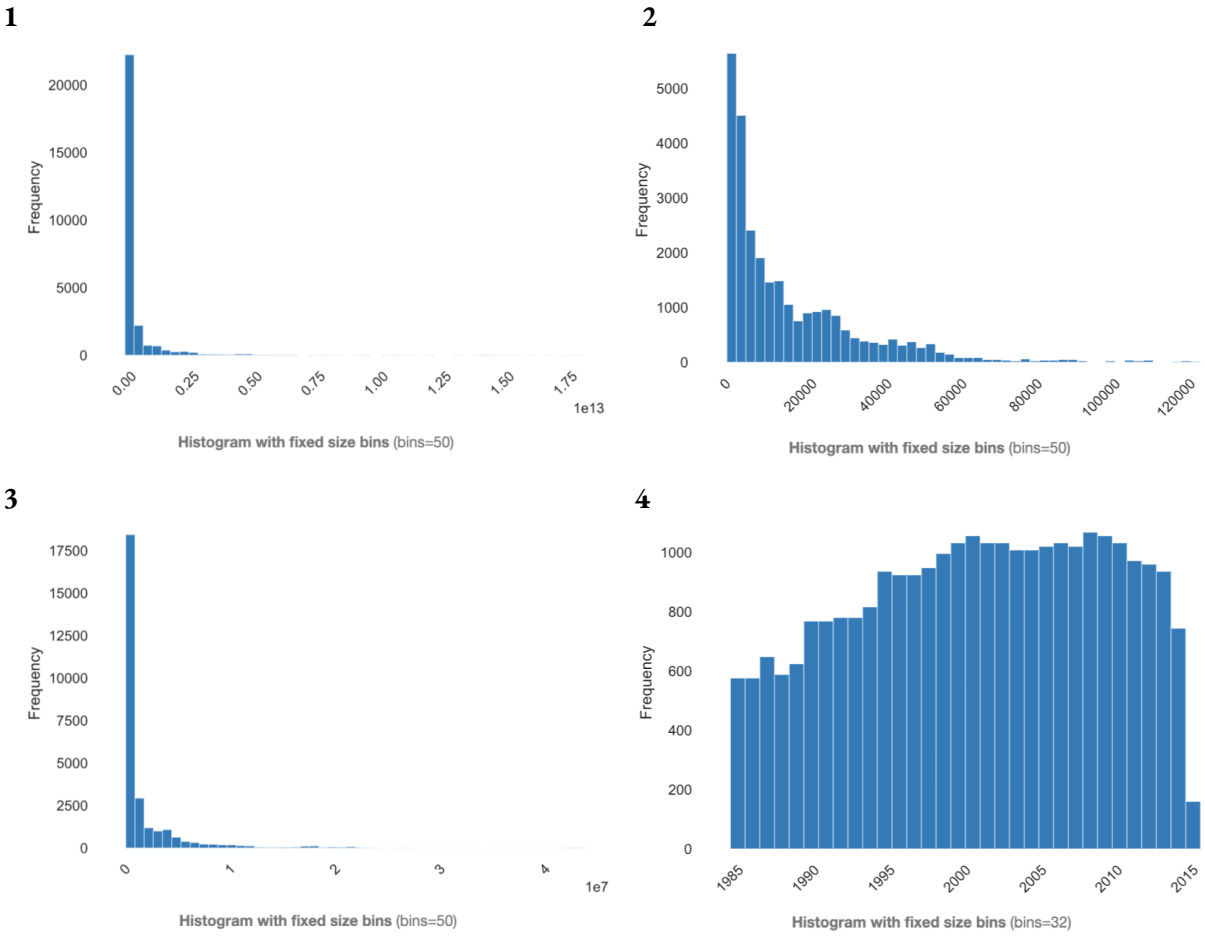


Figure 7: Histogram plots for (1) gdp_for_year, (2) gdp_per_capita, (3) population and (4) year for suicide-rates dataset

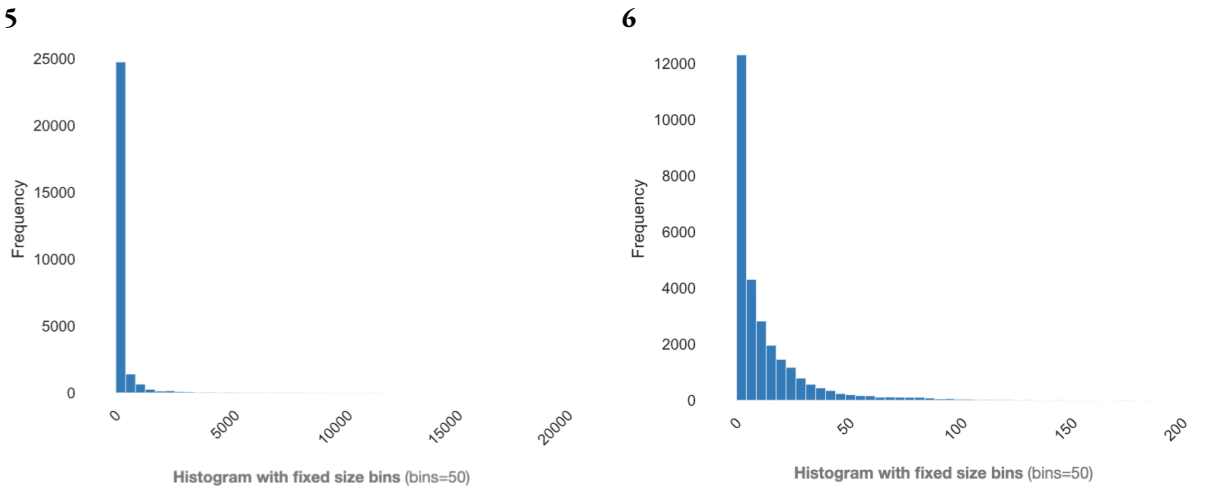
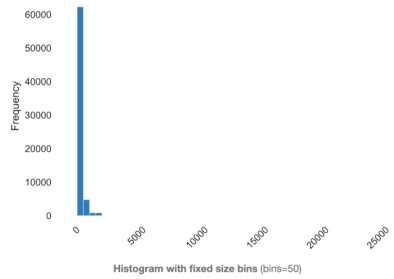
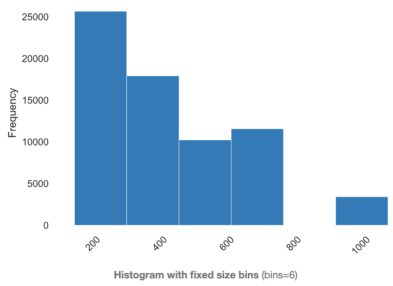


Figure 8: Histogram plots for (5) suicides_no and (6) suicides/100k pop for suicide-rates dataset

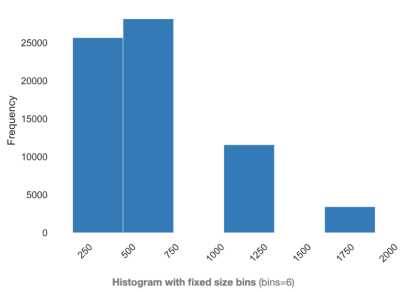
1



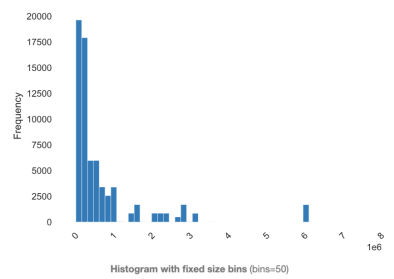
2



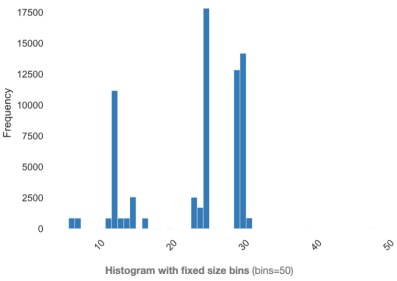
3



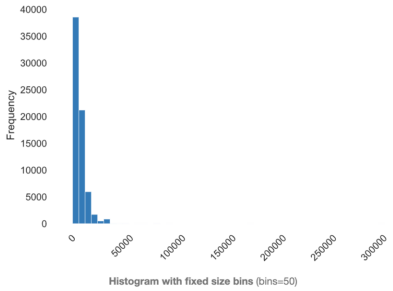
4



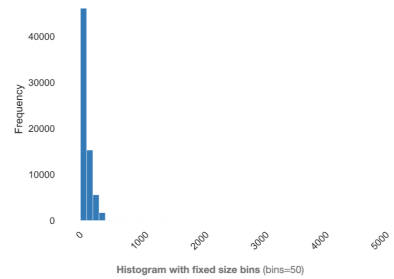
5



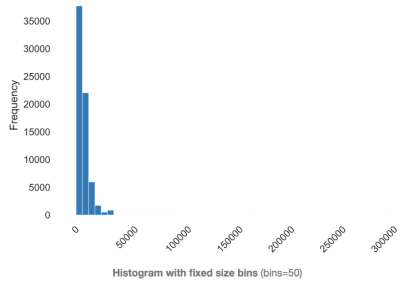
6



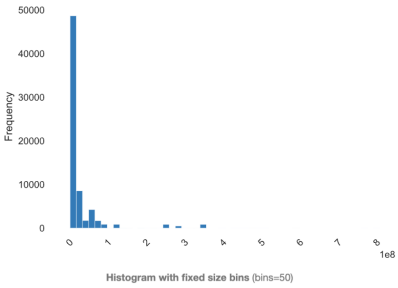
7



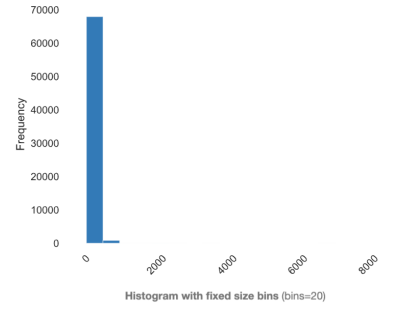
8



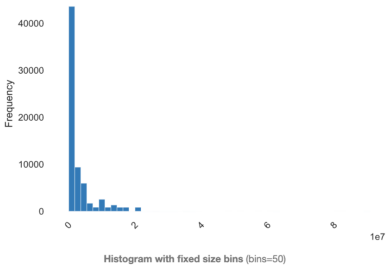
9



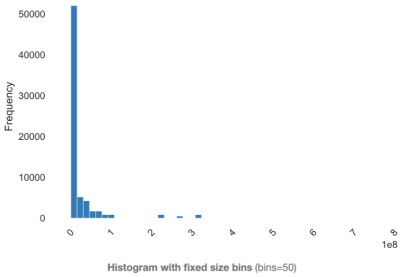
10



11



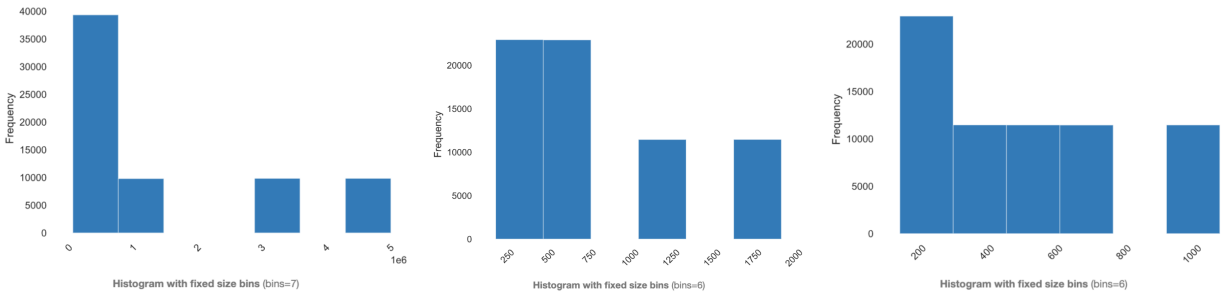
12



13

14

15



16

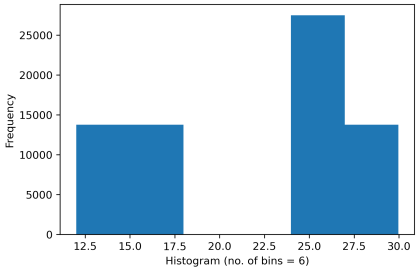
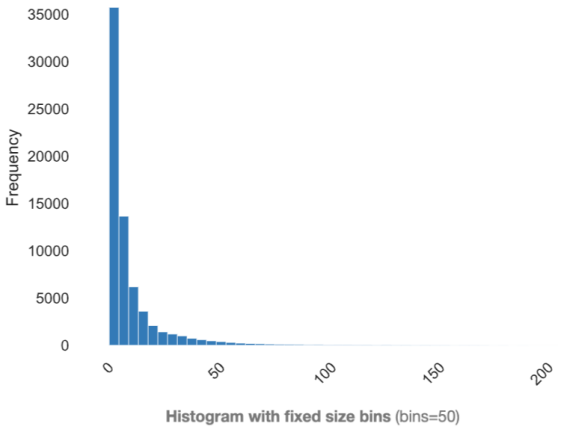


Figure 9: Histogram plots for (1) duration, (2) height, (3) width, (4) bitrate, (5) frame_rate, (6) frames, (7) i, (8) p, (9) size, (10) b, (11) i-size, (12) p-size, (13) o-bitrate, (14) o-width and (15) o-height (16) o-framerate for video-transcoding dataset. b-size has been omitted as it has only one value (all zeros), while plot (16) was generated manually.

17



18

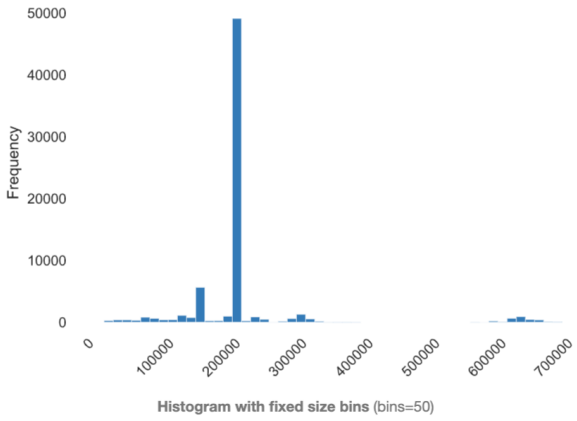


Figure 10: Histogram plots for (17) utime and (18) umem for video-transcoding dataset

Features with high skewness are not desirable in regression modelling because:

- Common parametric statistical tools, including regression analysis, require the distribution of the dependent variable to be Gaussian conditioned on the features and error component in the linear regression model. In other words, Gaussian approximation of parameter estimation is more likely to converge if the features follow a Gaussian distribution. Skewed features limit what kinds of valid statistical analysis one can perform on the said features.

- Skewed features provide misleading measures of location (moments), i.e., the mean cannot be used to represent the distribution of the features. Least-squares regression requires the mean to point to central tendency. In addition the tails of the distribution (from unbounded variance) act as outliers, affecting regression error metrics adversarially.
- Leads to polar width of confidence intervals.
- Parameter estimation on skewed distributions leads to disproportionate influence on the parameter estimates when minimized on squared error.

If a certain feature has high skewness, one can perform the following pre-processing techniques:

- Transformations:
    - Positive/right skewed distributions: Square root (weak), cube root (moderate), logarithmic (strong but cannot be applied to 0 values) or reciprocal (strong but cannot be applied to 0 values) transforms.
    - Negative/left skewed distributions: Power transforms (e.g., squares, cubes, box-cox etc.).
- Removing outliers (values with extreme distances from the mean of the distribution).
- Transformations coupled with normalization (e.g., z-score or min-max) and scaling or standardization.

Question 3:

In this question, we are asked to provide box-plots of categorical features for all three datasets against respective target variables. The valid categorical features and target variables for each dataset include:

- **Bike-sharing dataset**:
  - Categorical features - **season, yr, mnth, holiday, weekday, workingday** and **weathersit.**
  - Target variables - **casual**, **registered** and **cnt**.
- **Suicide-rates dataset**:
  - Categorical features - **sex, age,** and **generation**. Note that 'country' is also a categorical variable, but with high cardinality, hence we omit it from this analysis.
  - Target variables - **suicides_no** and **suicides/100k pop**.
- **Video-transcoding dataset**:
  - Categorical features - **codec** and **o_codec**.
  - Target variables - **umem** and **utime.**

Figure 11, 12 and 13 show the histogram plots of the categorical features for target variables 'casual', 'registered' and 'cnt' respectively for the bike-sharing dataset.

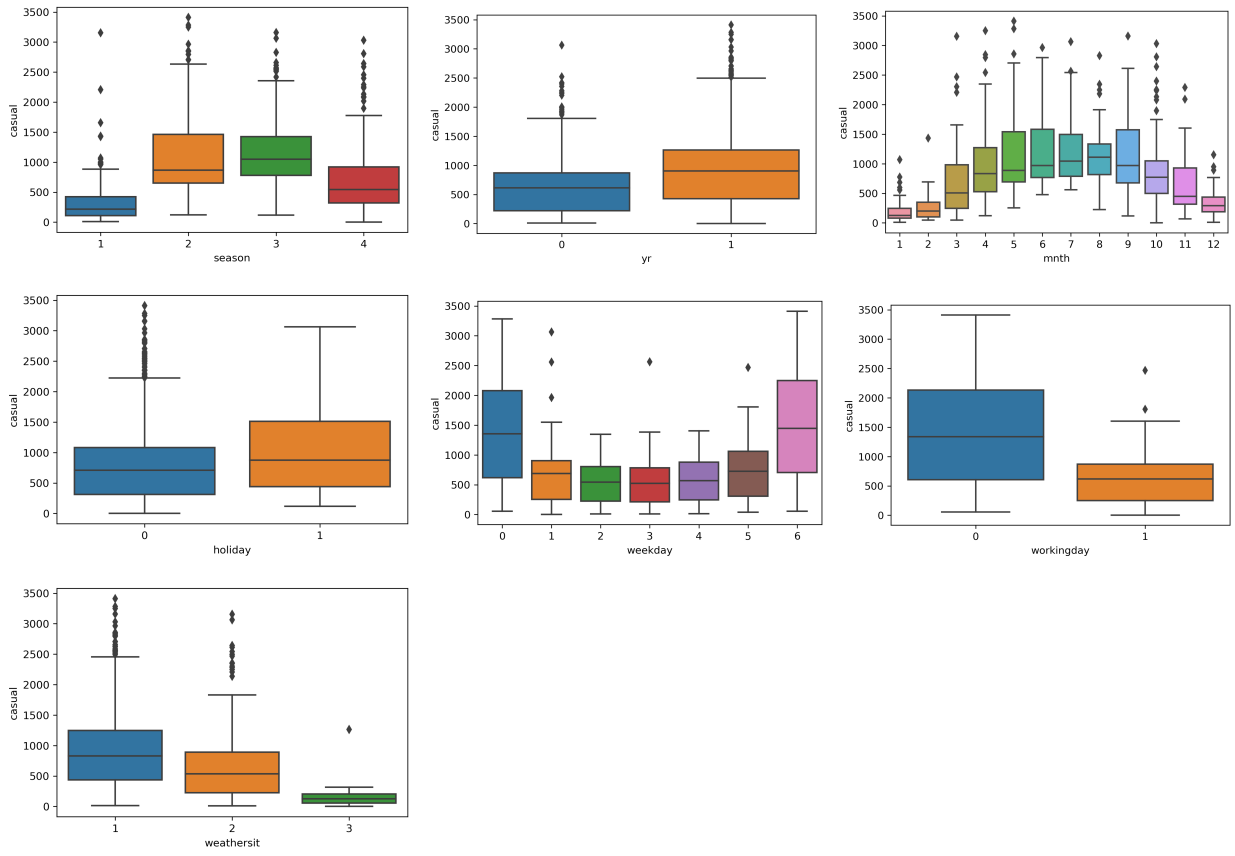


Figure 11: Box-plots for categorical features vs. 'casual' for bike-sharing dataset.

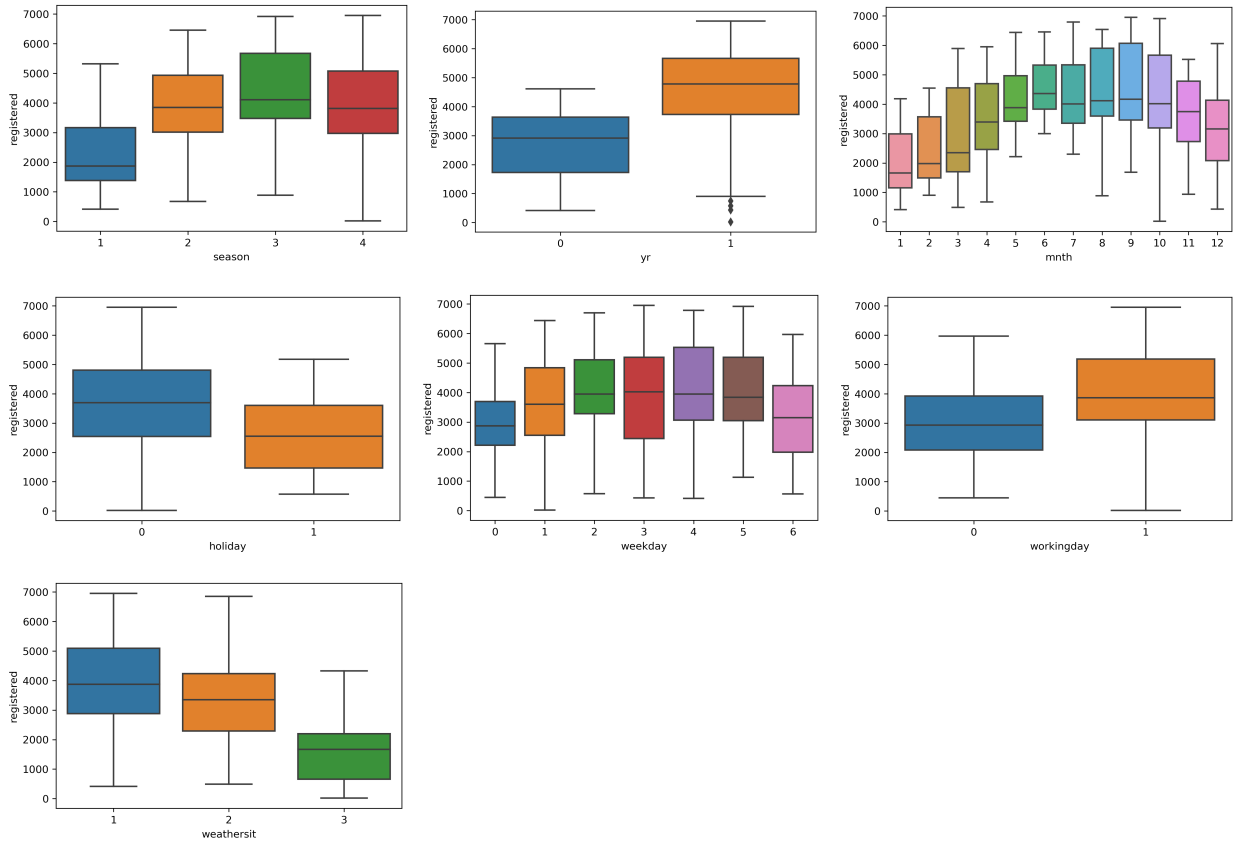
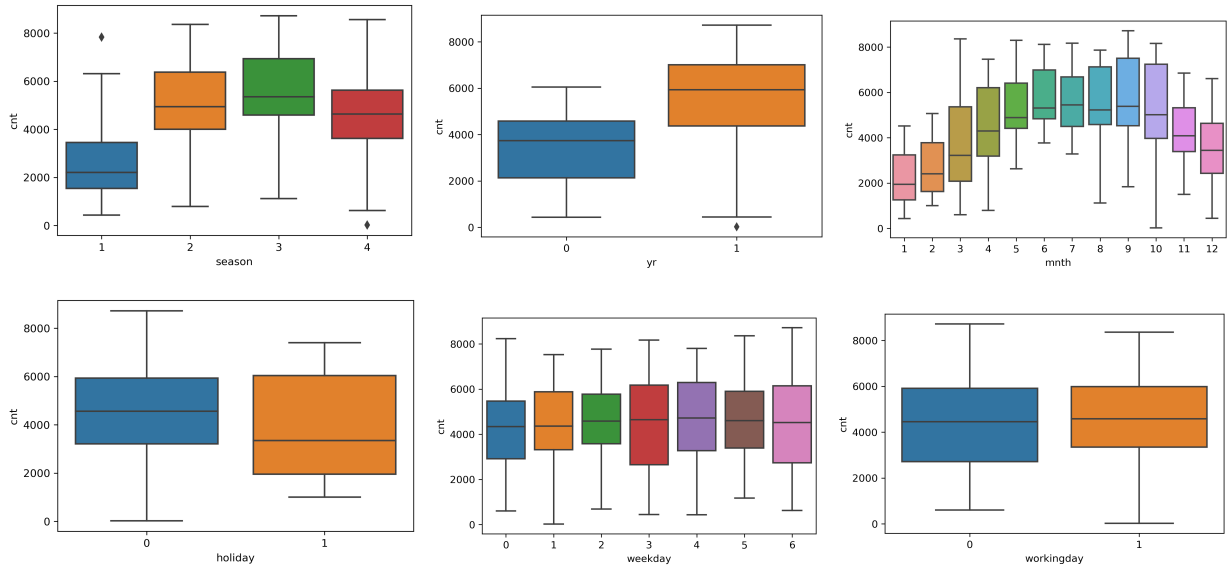


Figure 12: Box-plots for categorical features vs. 'registered' for bike-sharing dataset.



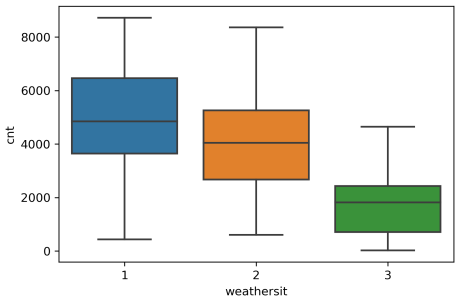


Figure 13: Box-plots for categorical features vs. 'cnt' for bike-sharing dataset.

We can make several inferences from the box-plots in Figures 11 to 13:

- Bike rental patterns are more predictable for each categorical feature for registered and overall cohort of users compared to casual riders. This is because of the presence of a large number of outliers for almost all of the categorical features for casual riders, while the presence of outliers is almost non-existent for registered and overall group of users. This is also reflected in Figure 6(5), which shows that the 'causal' target variable is right skewed and not normally distributed like 'registered' or 'cnt' variables (Figure 6(6), Figure 6(7)). This is not surprising, as casual users tend to be either tourists, visitors or leisurely riders from outside the city who do not follow regular bike rental patterns as registered users.
- Casual riders tend to rent bikes on holidays, whereas registered users tend to rent bikes on weekdays. As mentioned earlier, casual riders are mostly tourists or leisure travelers while registered users use bikes for commuting and work. This is also evident from the weekday and working day box plots, which shows a surge in bike rentals during the weekends and when it's not a working day by casual riders, whereas bike rentals are common on working days and middle of the week for registered users.
- More bikes are rented in summer and fall due to warmer and more comfortable temperatures during those seasons over winter and spring. This is reflected in the months during which bike rentals surge, especially May-October and also in the weather patterns plots, showing that people tend to rent bikes when the weather is clear or partly cloudy, supported by the heatmap plot in Figure 1, which shows that bike rentals for all three groups are positively correlated with the temperature and "real-feel" temperatures in the city. In addition, summer and the beginning of fall are both holiday seasons with people having more free-time than other seasons.
- For this particular company, the number of registered users is higher than the number of casual riders.
- Registered users are more likely to use the bike in extreme weather compared to casual riders, shown from the width of the box plots in 'weathersit' category.

Figure 14 and 15 show the histogram plots of the categorical features for target variables 'suicides_no' and 'suicides/100k pop' respectively for the suicide-rates dataset.

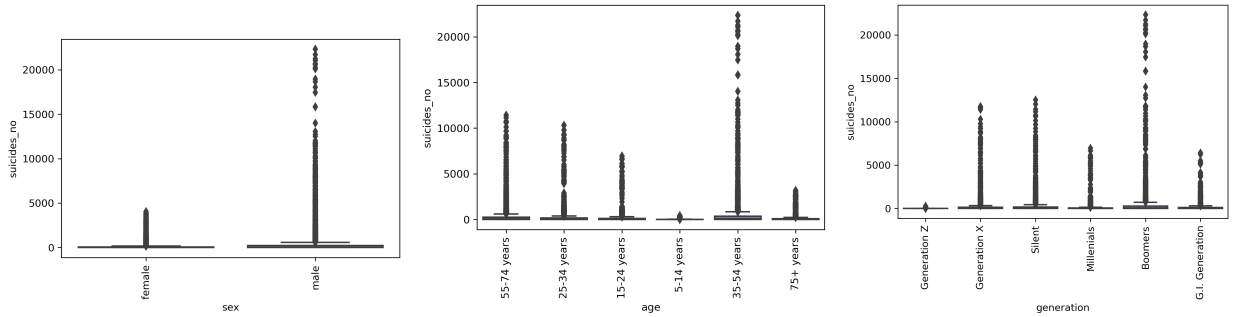


Figure 14: Box-plots for categorical features vs. 'suicides_no' for suicide-rates dataset.

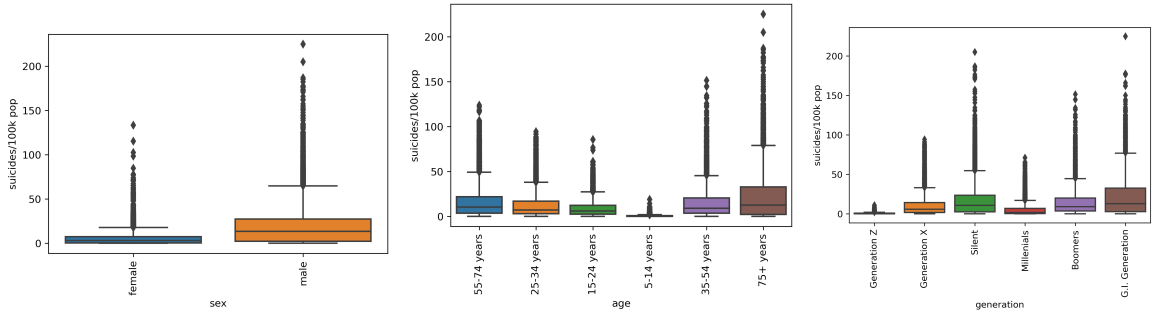


Figure 15: Box-plots for categorical features vs. 'suicides/100k pop' for suicide-rates dataset.

We can make several inferences from the box-plots in Figures 14 to 15:

- 'Suicides/100k pop' is a better parametric estimator for suicides rate over 'suicides_no' given the relative presence of outliers in both the target variables, in other words, normalized suicide count is much more Gaussian in nature over total number of suicides.
- Globally, men tend to commit more suicides than women.
- People aged 55-75+ are more likely to commit suicide than other age-groups. The lowest risk group is 5-14 years. In fact, the silent and greatest generations are at the highest risk, probably due to depression from isolation and being at the tailing-end of their careers. The other age-groups are either beginning their career, enjoying their childhood or are at their most productive career summits.

Figure 16 and 17 show the histogram plots of the categorical features for target variables 'umem' and 'utime' respectively for the video-transcoding dataset.

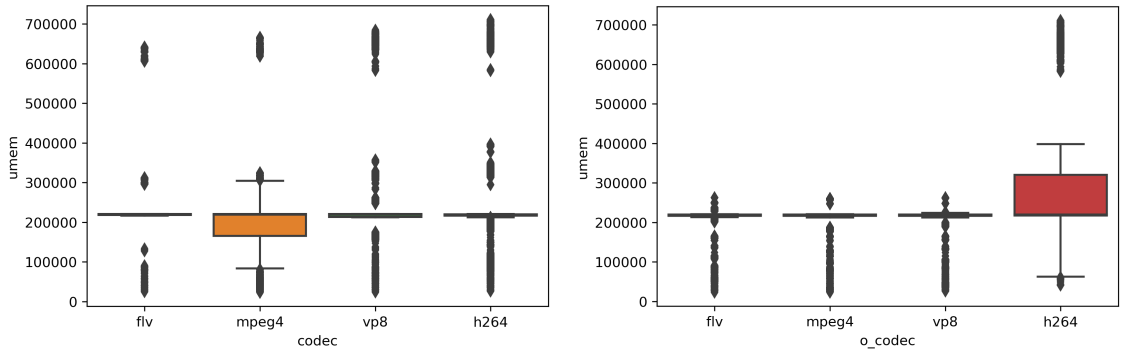


Figure 16: Box-plots for categorical features vs. 'umem' for video-transcoding dataset.

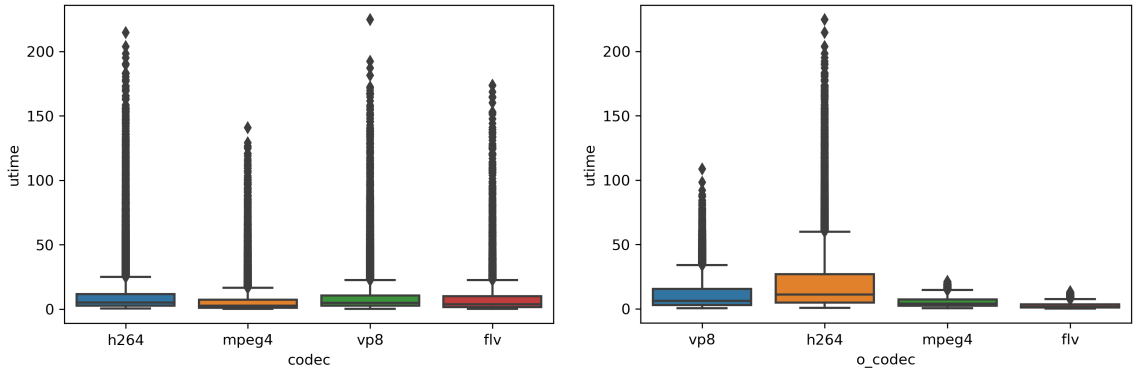


Figure 17: Box-plots for categorical features vs. 'utime' for video-transcoding dataset.

We can make several inferences from the box-plots in Figures 16 to 17:

- The mean of the distributions of 'umem' for both 'codec' and 'o_codec' are not significantly different. Hence, none of these categorical variables are suitable for estimating the total codec allocated memory for transcoding.
- For 'utime', 'o_codec' is suitable for estimating the transcoding time, as the mean of the distributions for the four classes are significantly different. However, 'codec' is not suitable for regression analysis, as the means are very close with significant presence of outliers (right skewed). This is evident from the heatmaps in Figure 4 as well, which show that 'o_codec' is correlated with 'utime'.
- H264 requires the most transcoding time, while FLV requires the least transcoding time.

Question 4:

Figure 18 shows the count number per day for a few months (January to June) in the year 2011 for bike-sharing dataset.

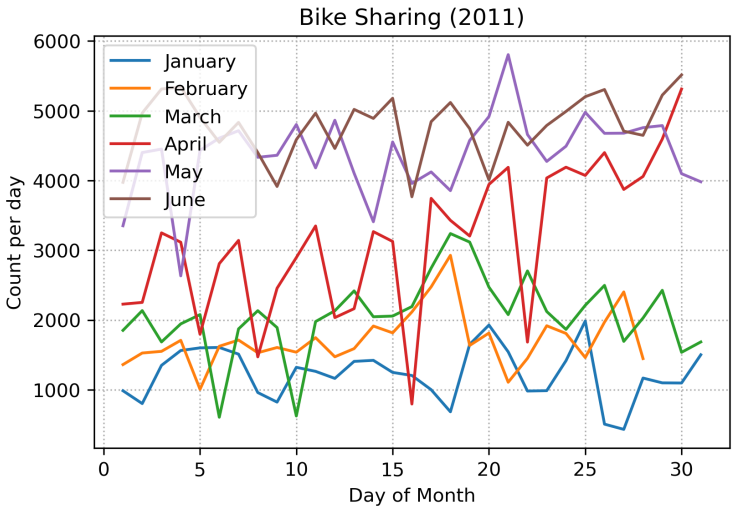


Figure 18: Plots for count number per day for bike rentals (2011) from January to June

We see three patterns in the plots:

- The highest number of rentals happens approximately around the 17-22nd of each month.
- There's a 3-5 day surge in bike rentals, followed by a 1-2 day drop in bike rentals. A quick look at the 2011 calendar shows the drop occurs during weekends/holidays and the peak rentals happen during weekdays.
- Bike-rentals at the end of the month are usually higher than the start of the month.

Question 5:

Figure 19 shows the normalized suicide rates ('suicides/100k pop') against time for different age groups and gender for countries with the longest timespan of records. The selected countries were 'Antigua and Barbuda', 'Argentina', 'Australia', 'Austria', 'Belgium', 'Grenada', 'Iceland', 'Mauritius', 'Netherlands' and 'Thailand', all of which had between 30 and 31 years worth of data. Note that the chosen countries will be slightly different for different students depending on how they break ties for the countries which have 30 years of data.

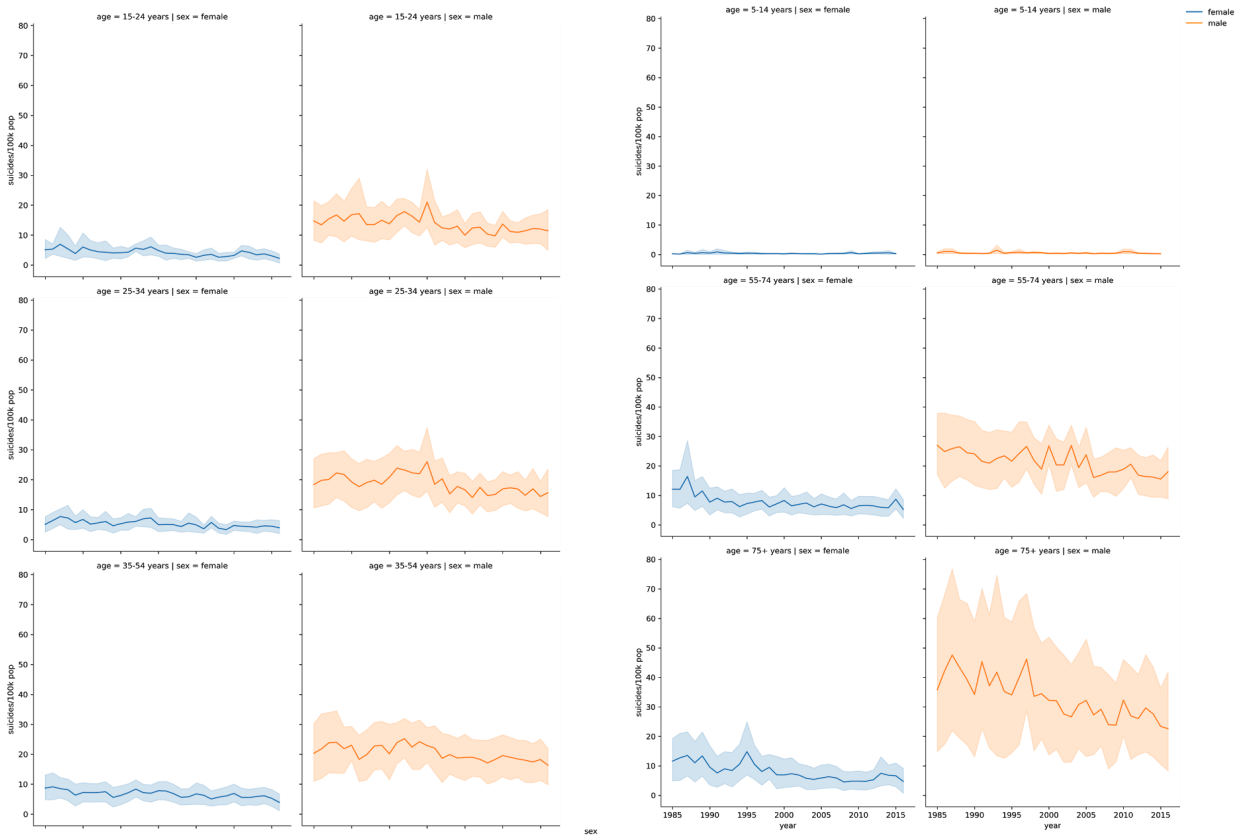


Figure 19: Plots for the normalized suicide rates ('suicides/100k pop') against time for different age groups and gender (generated using `seaborn.relplot`) for top 10 countries with highest record timespan.

We can make several observations from Figure 19:

- For all age groups except 5-14 years and gender, the suicide rate has reduced as the years progressed. The suicide rate for age group 5-14 years has remained relatively stable, neither increasing nor decreasing in the recorded time-frame.
- For all age groups except 5-14 years, men are more likely to commit suicide than women. The tendency is equal for age-group 5-14 years. The overall tendency is also shown in Figure 20 (left).
- With the exception of age group 5-14 years, the uncertainty in average suicide rate is higher for men that women. This is also evident from box-plots in Figure 14 and 15, which shows that the range of suicide

rates and quantiles for men are not only wider but the outlier count is also significantly higher than women.

- The highest risk groups are 55-74 and 75+ years. This is also shown in Figure 20 (right).
- The uncertainty bounds (presence of outliers) are greatest for the two oldest age-groups (55-74 and 75+ years). This is also evident from the box-plots shown in Figure 15.

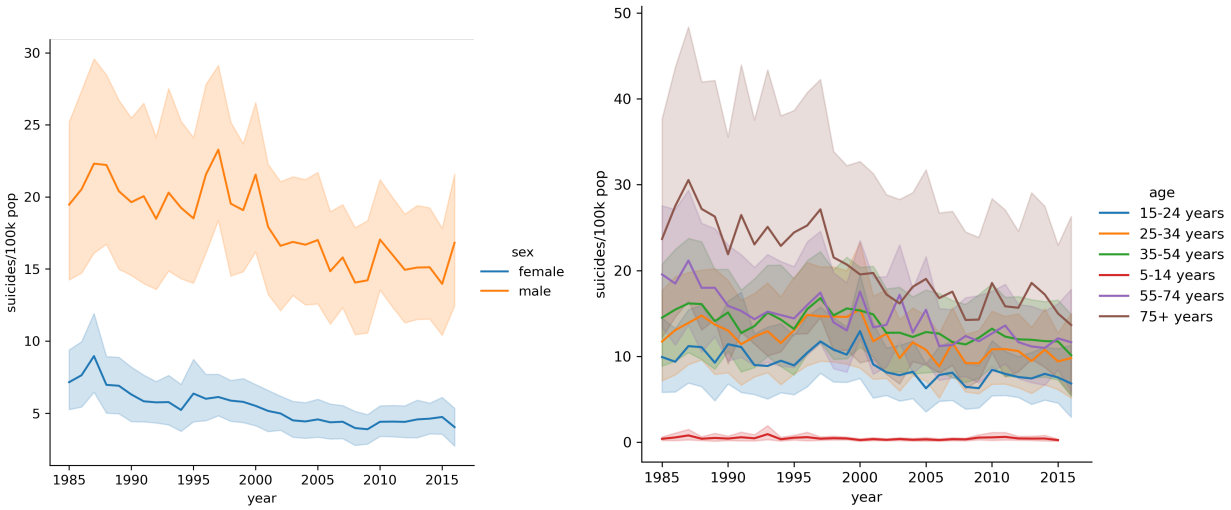


Figure 20: (Left): Plot for the normalized suicide rates ('suicides/100k pop') against time for different genders, (Right) Plot for the normalized suicide rates ('suicides/100k pop') against time for different age groups (generated using `seaborn.relplot`) for top 10 countries with highest record timespan.

Question 6:

Figure 21 shows the distribution of video transcoding times ('utime'), generated automatically using `pandas-profiling`. The mean of the video transcoding time is 9.996354821 and the median is 4.408.

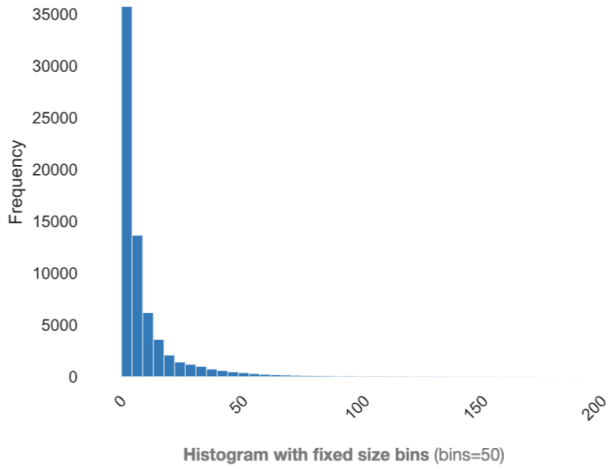


Figure 21: Distribution of video-transcoding times ('utime') for video-transcoding dataset

From Figure 21, we see that the distribution is positively/right skewed. This is also supported by the fact that the mean is greater than the median, which happens when a feature distribution is right skewed. Over half of the transcoding times (~36000 out of 68784 samples) are within the bin 0-5, while the majority of the transcoding times are within 0-50.

Question 7:

In scalar encoding (also known as label or ordinal encoding), we assign a unique integer to each label in a categorical variable. The problem with scalar encoding is that it imposes unwanted ordinality or rank in those categorical variables where the ordering of integers does not mean anything (e.g., name of countries) and assumes consecutive labels are uniformly and evenly distributed. In addition, scalar encoding can mislead the model to assume false natural ordering in the labels and capture illogical mathematical relationships among the labels (e.g., cat+dog = mouse, if cat = 1, dog = 2 and mouse = 3) that have no semantic meaning. For using scalar encoding, there must strictly be ordering or rank among the labels with similar qualitative difference (e.g., junior kindergarten, senior kindergarten, primary school and high school or Poor, Fair, Typical, Good, Excellent are evenly spaced with increasing rank) among two consecutive labels. However, when the choice of algorithms are decision trees or random forests, it is fine to use scalar encoding as tree-based algorithms can implicitly handle categorical variables regardless of encoding type.

On the other hand, in one-hot encoding (also known as dummy variable encoding when the redundant variable is dropped), each label is assigned a dummy variable that can take the value of either 1 or 0, i.e. each unique label in the category is added to the dataset as a feature in an orthogonal fashion. As a result, one-hot encoding provides equal importance to all the unique labels and discards any ordering or rank that may or may not have been present in the original categorical feature. There are three disadvantages of one-hot encoding:

- If ordering is present in the categorical variable, such ordering information is discarded. Hence, one-hot encoding is not recommended for ordinal variables.
- For highly cardinal variables (e.g., 'countries' in suicide-rates dataset), the feature space can blow up quickly if one-hot encoding is directly used, leading to 'Curse of Dimensionality'.
- Multicollinearity, redundancy and singularity: The dummy variable trap leads to dependency among the dummy variables when *n* variables are formed from *n* unique labels. Instead, it is recommended to use *n-1* variables. In addition, for the feature matrix to be invertible and be a full-rank parameterization in ordinary least-squares linear regression, it is recommended to use *n-1* variables instead of *n* variables, dropping the redundancy.

For bike-sharing dataset, the categorical features are 'season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday' and 'weathersit'. We observed that 'yr', 'holiday' and 'workingday' are already one-hot-encoded to *n-1* variables (i.e., these variables either take the value of 0 or 1, and according to the multicollinearity, redundancy and singularity explanation, they are already encoded properly with no redundancy using just 1 variable which takes the value of either 0 or 1). We used `pd.get_dummies()` from Pandas to convert 'season', 'mnth', 'weekday' and 'weathersit' to one-hot encoded variables.

For suicide-rates dataset, the categorical features are 'country', 'sex', 'age', and 'generation'. Since 'country' has a high-cardinality, as instructed in the question, we assigned the 101 unique countries to 6 continents (namely Asia, Europe, Oceania, South America, North America and Africa) automatically using

pycountry_convert library. We then one-hot encoded ‘continents’, ‘sex’, ‘age’ and ‘generation’ using pd.get_dummies()

For video-transcoding dataset, the categorical features are ‘codec’ and ‘o_codec’, which we converted to one-hot encoded variables using pd.get_dummies().

Question 8:

For standardizing the training sets for all three datasets, we used `StandardScaler()` function from SciKit-Learn, using fit and transform together on the training sets to mitigate skewness of the features by transforming the features to have zero mean and unit variance and hence look like Gaussian or normally distributed features. This ensures that all parameters contribute proportionally to the estimation step when minimized on squared error with a high confidence interval. In addition, standardization ensures that the measures of location (moments) points to central tendency, with the resulting distribution having a bounded tail (univariance). Mathematically, standard scaling subtracts the feature column from its mean divided by the standard deviation of the feature.

$$Z_X = \frac{X - \mu_X}{\sigma_X}$$

Feature scaling is desirable due to the following reasons:

- Reduces sensitivity of training to outliers and scales of features, leading to homogeneous distribution of weights across features.
- Improves convergence speed during model training by making the gradient-plane well-conditioned.
- Removes adverse effects of biased regularization term on smaller-scale features with larger weights (discussed in details in Question 12)
- Leads to consistent and predictable results across models.

Question 9:

In this question, we are asked to explore how feature selection affects model performance. For this question, we test "mutual information regression" (MI) and "F-regression) (F-Score) feature selection schemes for linear regression without regularization, linear regression with Lasso regularization and linear regression with Ridge regularization using default value of hyperparameters for all three regressors on all three datasets without standardization. We use `SelectKBest()` function to select the *k* best features for *k* ranging from 1 to maximum number of features in each dataset for each of the two feature selection schemes, and obtain average negative test root-mean-squared error (RMSE) (**the more positive the "negative test RMSE" is, the better**) from 10-fold cross validation. The target variables are 'cnt', 'suicides/100k pop' and 'utime' for bike-sharing, suicide-rates and video-transcoding datasets. Figure 22 to 24 shows the plots for the effect of feature selection on all three datasets. Note that we evaluate mutual information score only on bike-sharing dataset, as it takes a long time to converge and often causes out-of-memory errors on the video-transcoding and suicide-rates dataset due to larger number of samples and more number of features.

MI (or information gain) is defined as a non-parametric non-negative measure of dependency between two features, which is equal to 0 if two features are independent and positive depending on how dependent the features are. MI is also known as the expected value of the pointwise mutual information. The intuition is that features that are correlated with the target variables should contain large amounts of Shannon information about the dependent variable. Mathematically:

$$I(X;Y) = D_{KL}(P_{(X,Y)}||P_X \otimes P_Y)$$

When the joint distribution is parallel to the product of the marginal distributions in the space of probability distributions (measured using Kullback-Leibler divergence), then MI is 0, indicating X and Y are independent.

F-Score performs univariate linear regression tests, i.e., F-tests to evaluate the significance level of the improvement in performance of the model with respect to the addition of new variables. It tells us if a complex model with all the features of a simpler model is significantly better than the latter with respect to the p-value (measures the individual effects of multiple regressors). Mathematically:

$$C = \frac{(X_i - \mu_{X_i}) \times (y - \mu_y)}{\sigma_{X_i} \times \sigma_y}$$

C is then converted to an F-score and then to a p-value. In other words, F-Score estimates the degree of linear dependency between the entropy of two random variables.

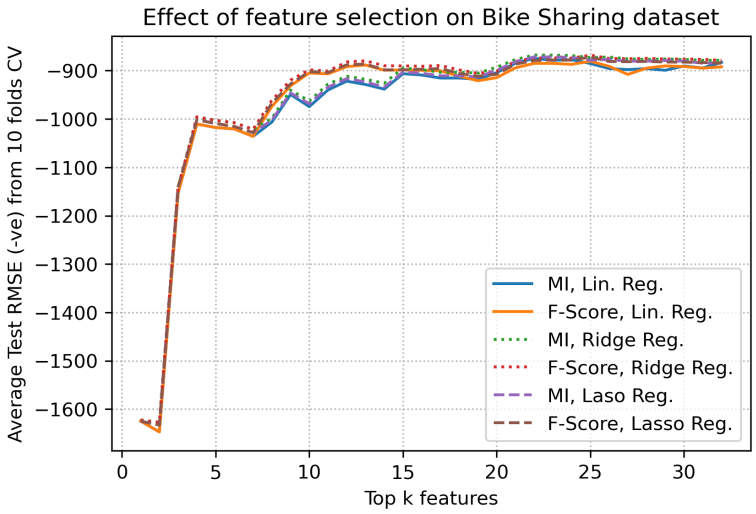


Figure 22: Effect of feature selection on test RMSE for bike-sharing dataset.

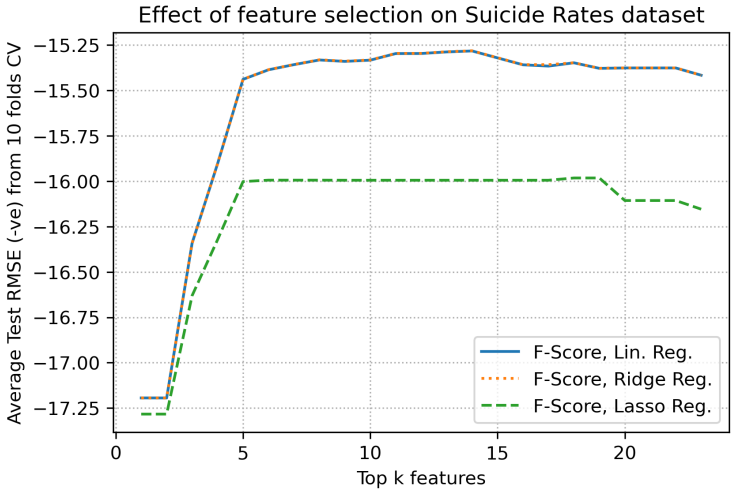


Figure 23: Effect of feature selection on test RMSE for suicide-rates dataset.

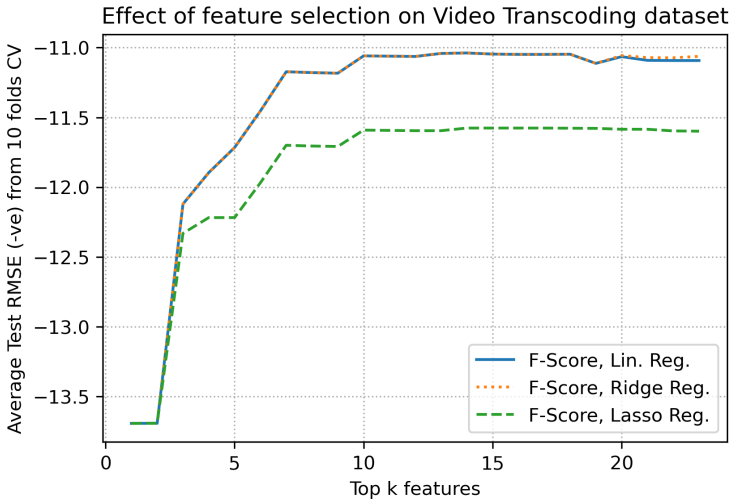


Figure 24: Effect of feature selection on test RMSE for video-transcoding dataset.

We can make several inferences from the plots in Figure 22 to 24:

- For all three models (linear, lasso and ridge regression), as the number of features increases, the test RMSE improves and converges to a constant value when a certain number of features are selected. This is because all of the information-conveying features that have the maximum correlation with the target variable have already been selected. For all three datasets, a good value for the number of features is 10. Beyond this value, there's a possibility of overfitting and we should avoid selecting more features. Feature selection helps reduce complexity and improve interpretability and generalizability of chosen models.
- While MI is able to capture non-linear relationships between features and target variables and provides information on the combined effect of features (both of where F-score fails), MI takes more samples to provide the best features and is much slower than F-Score. Furthermore, MI is unbounded (not normalized) and has a much higher memory count compared to F-Score as MI expects discretized or binned samples.
- For the bike-sharing dataset, both F-Score and MI converge to similar test-RMSE when the number of selected features approaches the total number of available features in the dataset. For a smaller number of features (e.g., 10), F-Score outperforms MI for all three models. We hypothesize that this happens because of the low number of samples in the bike-sharing dataset as well as continuous nature of the salient features (e.g., 'atemp', 'windspeed', 'temp' are all continuous), which MI does not pick initially as MI expects discrete features or binned samples.

Question 10 to 13:

In questions 10 to 13, we are asked to compare the performance of linear, lasso and ridge regression models on the three datasets, along with the effect of feature scaling and meaning of p-value.

Objective functions for each regression model:

- Ordinary least squares:

$$J(W) = \frac{1}{2N}\sum_{i=1}^{M}(y_i - \sum_{j=0}^{P} w_j x_{ij})^2$$

- Lasso regression:

$$J(W) = \frac{1}{2N}\sum_{i=1}^{M}(y_i - \sum_{j=0}^{P} w_j x_{ij})^2 + \lambda \sum_{j=0}^{P} |w_j|$$

- Ridge regression:

$$J(W) = \frac{1}{2N}\sum_{i=1}^{M}(y_i - \sum_{j=0}^{P} w_j x_{ij})^2 + \lambda \sum_{j=0}^{P} w_j^2$$

y is the target variable x represents the features and w represents the weights assigned to each feature. λ represents regularization strength.

How each regularization scheme affects the learned hypotheses (Question 11):

Lasso regression is also known as linear regression with L1 regularization, while ridge regression is also known as linear regression with L2 or Tikhonov regularization. Regularization encourages generalization by yielding simpler models to prevent overfitting. Regularization attempts to improve generalizability by increasing estimator bias while reducing variance, causing many of the weights to be very small, leading to a simpler model which performs well on unseen test data but may have lower overall RMSE.

- For both kinds of regularization, as regularization strength λ increases, more weights in the learned model are set to 0. This is due to the fact that the L1 and L2 regularization are formulated as minimizing the weights. Thus, to prevent the regularization portion of the least squares equation from becoming too large, the optimization program will attempt to minimize the weights, ideally the weights should approach 0. However, as models become more and more simpler, they start losing the most important weights required promoting the salient features to contribute to the final estimation. As a result, very large values of regularization strength will cause the test accuracy to drop dramatically (underfitting). However, finite values of regularization strength can help make the model more generalizable on unseen test data, preventing overfitting. Regularization also makes the model more stable by reducing variance and sensitivity to outliers and increasing bias.

- Lasso regularization is suitable for feature selection and construction of simple and sparse linear regression models, where features associated with 0 weights can be discarded. This is because L1 regularization encourages all kinds of weights to shrink to 0, regardless of size of w, as the subgradient of $||\mathbf{w}||$ is sign($\mathbf{w}$). L1 regularization is more likely to zero out coefficients than L2 regularization for similar test accuracies because it assumes priors on the weights sampled from an isotropic Laplace distribution (linear descent), which has a much lower Q factor than Gaussian distribution (exponential descent). Thus, only a sub-selection of features are active in the learned hypothesis.
- Ridge regularization, which assumes priors on the weights sampled from a unit Gaussian distribution, is suitable for reducing the effects of collinear features, which can lead to increased variance (hence instability and sensitivity to outliers) of the model. This is because the subgradient of $||\mathbf{w}||^2$ depends on not just the sign of $\mathbf{w}$ but also the magnitude of $\mathbf{w}$, which can be thought of as scaling the variance of weights, reducing dependence of the model on few features and encouraging distributed contribution of all the features. This leads to regression models with diffuse weights compared to sparse weights from L1 regularization. Thus, ridge regularization promotions participation of all the features in the learned hypothesis to prevent overfitting.

Best regularization scheme with optimal penalty parameter (Question 12):

For this question, we test the performance in terms of train and test RMSE (average negative test RMSE, higher is better) of linear regression, lasso regression and ridge regression on top 10 selected features in the bike-sharing, suicide-rates and video-transcoding dataset. For bike-sharing dataset, we test top 10 features selected via both MI and F-Score, while for the other two datasets, we use the top features selected via F-Score. To compute the optimal penalty parameter for each regularization scheme as well as test general model performance, we perform grid search with 10-fold cross validation (`GridSearchCV()`), with λ in the range of $[10^{-4}, 10^{4}]$. It is also possible to use `LassoCV()` and `GridCV()` to calculate the optimal penalty terms. We also test the effects of feature scaling / standardization, which we explain in the next question.

Table 1 reports the best penalty parameters obtained for each possible choice of configurations and models for the three datasets, along with average test and train RMSE for best penalty parameters obtained via 10-fold grid search cross-validation.

- For bike-sharing dataset, the best test RMSE is obtained via ridge-regression, with an average test RMSE of -878.1469958606837 on features selected via F1-regression with an optimal penalty parameter of 100.0 and standardization/feature scaling enabled.
- For suicide-rates dataset, the best test RMSE is obtained via ridge-regression, with an average test RMSE of -15.322322304708754 on features selected via F1-regression with an optimal penalty parameter of 1000.0 and standardization/feature scaling enabled.
- For video-transcoding dataset, the best test RMSE is obtained via lasso-regression, with an average test RMSE of -11.054351412935565 on features selected via F1-regression with an optimal penalty parameter of 0.1 and standardization/feature scaling enabled.

Table 1. Performance summary of linear, lasso and ridge regression on bike-sharing, suicide-rates and video-trancoding datasets for top 10 features (best test RMSE highlighted in green for each dataset).

Dataset	Model	Standardization / Scaling	Feature Selection Scheme (top 10 features)	Optimal penalty parameter	Train RMSE (negative, higher is better)	Test RMSE (negative, higher is better)
Bike sharing	Linear regression	Yes	Mutual Information	-	-871.9979472259711	-974.4892680662463
			F1- Regression	-	-836.3621038950808	-904.875354514855
		No	Mutual Information	-	-871.9979472259711	-974.4892680662463
			F1- Regression	-	-836.3621038950808	-904.875354514855
	Lasso regression	Yes	Mutual Information	100.0	-871.9979499730189	-946.5350865872509
			F1- Regression	10.0	-836.3621040459991	-897.4951623793668
		No	Mutual Information	10.0	-836.3621040837518	-952.3265864115558
			F1- Regression	10.0	-836.3621040837518	-899.5074463695244
	Ridge regression	Yes	Mutual Information	100.0	-871.9979472261564	-935.7339113922999
			F1- Regression	100.0	-836.3621038954203	-878.1469958606837
		No	Mutual Information	10.0	-871.9979474460961	-962.7636703128888
			F1- Regression	10.0	-836.3621041845709	-892.70230942667
Suicide rates	Linear regression	Yes	F1-Regression	-	-15.420589566680139	-15.332744465450682
		No		-	-15.42061968119477	--15.333207286266429
	Lasso regression	Yes		0.1	-15.42018215325216	-15.32750979337663
		No		0.01	-15.420182168435108	-15.330688655388517
	Ridge regression	Yes		1000.0	-15.42018215133271	-15.322322304708754
		No		1000.0	-15.42018215133271	-15.32570785324082
Video transcoding	Linear regression	Yes	F1-Regression	-	-11.038710923489962	-11.059454228951008
		No		-	-11.038710923489962	-11.059454228951019
	Lasso regression	Yes		0.1	-11.038711188883545	-11.054351412935565
		No		0.01	-11.03871093075142	-11.059330236660127
	Ridge regression	Yes		100.0	-11.038710923489962	-11.059033334149994
		No		100.0	-11.038710923489962	-11.059236787276069

Does feature scaling play any role? (Question 12)

From Table 1, we can see that feature scaling does not make any significant difference in the average test or train RMSE when regularization is not used. However, feature scaling improves the test RMSE when regularization is used.

Feature scaling does not cause any changes in the original dataset distribution. Without regularization, the change in values caused by feature scaling will be reflected by corresponding changes in the weights of the model to achieve the lowest RMSE. Since the data distribution is the same as before, the changes caused by feature scaling will reflect linearly on the changes in the coefficients, yielding no performance gains or losses. As a result, we see no change in test RMSE with or without feature scaling when regularization is absent.

Now, consider two salient features, one which has a much smaller range and absolute value than the other one. Without normalization, the coefficients of these two features will be unbalanced, with the smaller feature having a larger coefficient than the larger feature in order to have a balanced outcome on the target variable. Since regularization aims at minimizing the weights of the regression model, it would remove or penalize the coefficients of the smaller feature, while having negligible effect on the weights of the larger feature. Feature scaling ensures that the weights assigned to each feature does not get adversely affected by a "biased regularizer penalizing smaller features" when regularization is used and leads to more stable and well-conditioned models. In addition, standardization also improves model convergence speed by making training less sensitive to scale of features.

Meaning of p-values (Question 13):

The p-value for each feature tests the null hypothesis that the feature has no correlation with the target variable (probability of the weights of the feature being 0) at the population level. A feature with a high p-value (> 0.05) indicates that there is insufficient evidence to find any meaningful correlation of that feature with the changes in the target-variable. On the other hand, a feature with a low p-value (< 0.05) indicates that the probability of the coefficients of that particular feature being 0 is low, which indicates that the feature is well-suited as a salient feature, contributing to changes in the target variable and ultimately rejecting the null hypothesis for that particular predictor (statistically significant feature). Thus, the most significant features will have a small p-value. For example, when applying `statsmodels.regression.linear model.OLS()` to the suicide-rates dataset, we get the following list of p-values:

```
gdp_per_capita ($)            8.766941e-48
 gdp_for_year ($)             1.003279e-11
age_75+ years                 1.078009e-06
Continent_EU                  4.850388e-06
age_55-74 years               7.534267e-05
sex_male                      9.415642e-05
```

```
generation_G.I. Generation    1.003124e-04
age_35-54 years               1.274671e-04
Continent_OC                  1.912970e-04
Continent_AS                  3.764538e-04
generation_Boomers            3.824482e-04
generation_Generation Z       4.133131e-04
generation_Silent             4.412699e-04
age_25-34 years               4.422605e-04
Continent_SA                  4.666982e-04
generation_Generation X       4.824022e-04
generation_Millenials         6.498364e-04
year                          1.026107e-03
sex_female                    1.279165e-03
age_15-24 years               1.978574e-03
Continent_AF                  2.456121e-03
Continent_NA                  2.921954e-03
population                    7.801250e-03
age_5-14 years                6.607614e-02
```

The top features include ‘gdp_per_capita’, ‘gdp_for_year’, ‘age’, ‘continent’ and ‘sex’, which partially matches the salient features obtained from the heatmap in Question 1. The slight difference is probably due to the influence of one-hot-encoded categorical variables, which increased the influence of numerical variables such as gdp over categorical variables.

Question 14 to 16:

In questions 14 to 16, we are asked to compare the performance of various degrees of polynomial regression on the three datasets, along with identifying the most salient polynomial features and the effect of inverse features.

Finding the most salient features (Question 14):

The results found in this question relate to the best polynomial regressors found via 10-fold grid-search cross-validation for each dataset in Question 15 (grader is urged to look at Question 15 first before Question 14). To find the most salient features, we find the coefficients with the highest absolute values in the polynomial model. The most salient (top 5) features for the best-performing polynomial regressors for each of the three datasets are:

- Bike-sharing dataset: 'temp', 'atemp', 'windspeed', 'atemp^2', 'temp atemp'.
- Suicide-rates dataset: 'sex_female', 'sex_male', 'sex_female^2', 'sex_female^3', 'sex_male^2'.
- Video-transcoding dataset: 'o_codec_flv', 'o_codec_flv^2', 'o_codec_h264^2', 'o_codec_h264', 'o_codec_mpeg4'.

For the bike-sharing dataset, we see that the most salient features are various mathematical combinations (e.g., squared, multiplied or individual features) of temperature, "real-feel" temperature and wind speed. This is supported by the heatmap we observed in Figure 1, where 'temp' and 'atemp' had the highest absolute correlations with 'cnt' target variable. As mentioned earlier, people tend to bike in warmer and more comfortable weather conditions. In addition, a higher wind speed during warm weather indicates a pleasant riding experience in the summer breeze, leading to more bike rentals.

For suicides dataset, we see that the most salient features are various mathematical combinations of sex. This is evident from the plots in Figure 19 and Figure 20 (Left), where it was shown that men tend to commit more suicides globally than women. This is also supported by the heatmap in Figure 3, where it was shown that 'sex' had the highest absolute correlation with 'suicides/100k pop' target variable, along with the box-plots in Figure 15.

For the video-transcoding dataset, the most salient features come directly from mathematical combinations of output codecs. This matches the observation in the heatmap in Figure 4, where it was shown that 'o_codec' had one of the highest absolute correlations along with 'o_height' and 'o_width', along with the box-plots in Figure 17. The output codec determines the compression and decompression time required for processing the video, with codecs such as mpeg4 providing higher compression rates than h264. In addition, codecs also determine the output screen size (height and width) of the video, with larger videos taking more transcoding time than smaller ones.

Finding the best polynomial degree (Question 15):

For this question, we test the performance in terms of train and test RMSE (average negative test RMSE, higher is better) of polynomial regression of various degrees on top 10 selected features in the bike-sharing, suicide-rates and video-transcoding dataset. Since F-Score selected features performed better than MI, we selected top 10 features from F-Score. For the model, we used ridge regression to incorporate appropriate regularization term (L2) to prevent overfitting, with the penalty parameter being optimized using 10-fold grid search cross-validation. The hyperparameter space for the grid search also included the degree of polynomial features, which ranged from 1 to 10 for bike-sharing dataset, 1 to 5 for suicide-rates dataset and 1 to 4 for video-transcoding dataset (we had to use different polynomial degree search spaces for each dataset because anything higher than the stated values caused Python to run out of 128 gigabytes of RAM in our GPU machine). To generate polynomial features, we used `PolynomialFeatures()` function SciKit-Learn within the pipeline. Figure 25, 26 and 27 shows the best average test and train RMSE versus the degree of polynomial for bike-sharing, suicide-rates and video-transcoding dataset respectively:

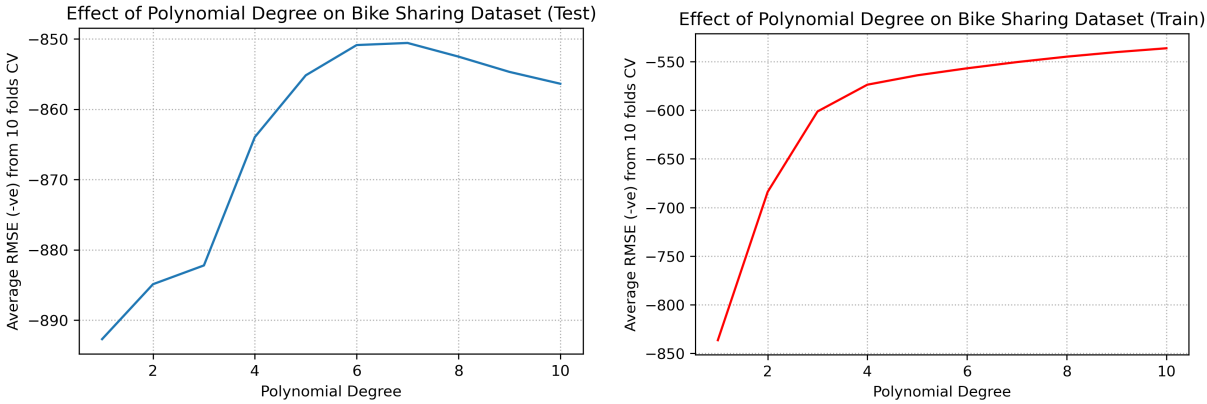


Figure 25: Effect of polynomial degree on bike sharing dataset (test and train).

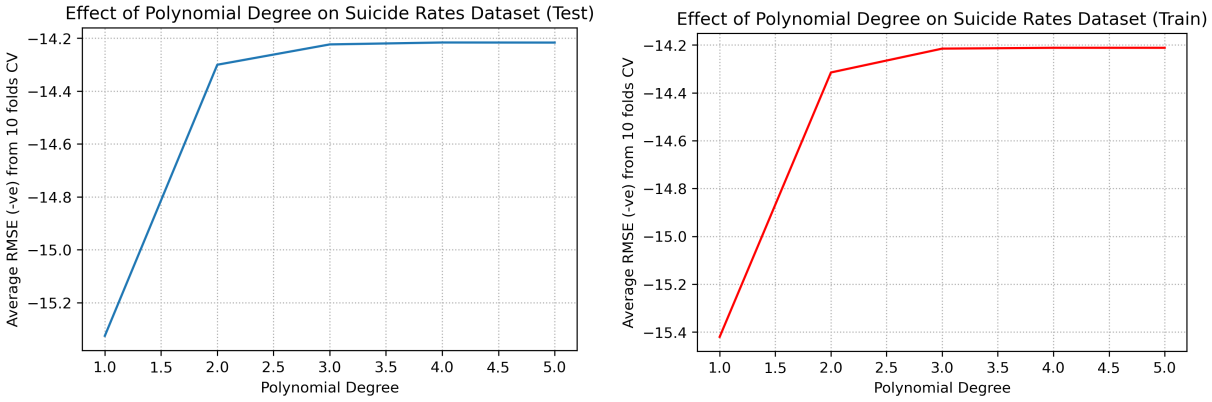


Figure 26: Effect of polynomial degree on suicide-rates dataset (test and train).

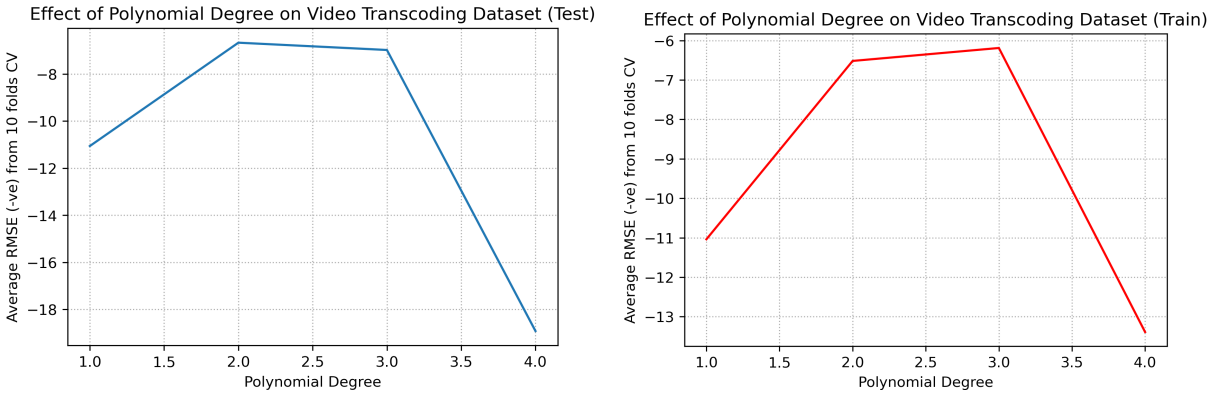


Figure 27: Effect of polynomial degree on video-transcoding dataset (test and train).

From Figures 25 to 27, we see that for the test RMSE, as polynomial degree increases, the RMSE improves until a particular threshold, beyond which the RMSE either stays constant or starts to become worse. Higher polynomial degrees give rise to larger models with more model capacity, however, beyond a critical point, the complexity of the model tend to cause overfitting (which is clear from the training graphs in Figure 25 and 26, which converges to a constant value, which the test RMSE curves start to drop). The best degree of polynomial for the 3 datasets (inferred from the maximum value of the average negative test RMSE), along with average train and test RMSE from 10-fold grid-search cross validation are as follows:

- Bike-sharing dataset: 7 (train RMSE: ~ -550, test RMSE: ~ -851)
- Suicide-rates dataset: 4 (train RMSE: ~ -14.20, test RMSE: ~ -14.22)
- Video-transcoding dataset: 2 (train RMSE: ~ -6.5, test RMSE: ~ -6.8)

We cannot increase the polynomial degree in an unconstrained manner because:

- Increasing polynomial degree causes exponential explosion of parameters needed to be stored in memory as well as compute requirements. In fact, we ran out of 128 gigabytes of memory when trying to increase the polynomial search for the suicide-rates and video-transcoding dataset. Mathematically, if we have n features and d degree polynomial, the number of features N generated is as follows::

  $$N(n, d) = C(n + d, d)$$

  where, C is the binomial coefficient.
- Although higher-degree polynomial models have a larger model capacity, higher polynomial degrees lead to a greater possible combination of features from the raw features, which can lead to overfitted, complex and uninterpretable models. In fact, if we look at the test RMSE in Figures 25 and 27, we observe that the test RMSE starts to become worse if the polynomial degree is increased beyond a certain threshold. As a result, we should only increase the polynomial degree upto a particular threshold and choose the degree beyond which the test RMSE does not change significantly or starts dropping. The search space should be small to ensure one does not run out of memory during grid search.

Inverse features (Question 16)

Inverse of particular features might yield a new feature that is highly correlated with the target variable, which would otherwise be impossible to craft using polynomial regression as it cannot yield inverse relationships (degree is always positive). For the video-transcoding dataset, an example of a feature whose inverse should in theory, perfectly correlate with the transcoding time 'utime' is the output bitrate or 'o_bitrate', as inverse of output bitrate provides a sense of transcoding time. In other words, output bitrate used for transcoding is inversely proportional to the transcoding time. Thus, the recipe for our inverse feature include:

```
Inv_feat = (o_height * o_width) / o_bitrate
```

We retrained the best model obtained in Question 16 (degree = 2 and associated optimal penalty parameter) for the video-transcoding dataset with the newly appended feature to the suicide-rates dataset. The obtained metrics are as follows:

- Average Test RMSE (-ve) without inverse feature (degree = 2): -6.669831310279045
- Average Test RMSE (-ve) with inverse feature (degree = 2): -6.063805812065415

We see that the addition of the new feature has caused significant improvements in the test RMSE and proves our hypothesis that the inverse of 'o_bitrate' is correlated with the video transcoding time.

Question 17 to 20:

In questions 17 to 20, we are asked to train a multilayer perceptron (MLP), systematically explore its hyperparameter space and activation functions and compare its performance with linear regression.

MLP versus linear regression (Question 17):

The results found in this question relate to the best MLP found via 10-fold grid-search cross-validation for each dataset in Question 18 (grader is urged to look at Question 18 first before Question 17), as well as the best linear regressors from Table 1. Table 2 shows the performance comparison of best-performing linear regression model and MLP model on bike-sharing, suicide-rates and video-transcoding datasets.

Table 2. Performance comparison of linear regression and MLP with most optimal hyperparameters on bike-sharing, suicide-rates and video-trancoding datasets for top 10 features

	Best Average Test RMSE (-ve, higher is better) (on top 10 features)		
Dataset	**Linear Regression (without regularization)**	**Linear Regression (with Lasso or Ridge regularization)**	**MLP**
Bike-sharing	-904.875354514855	-878.1469958606837	-859.001551074912
Suicide-rates	-15.332744465450682	-15.322322304708754	-14.10755903748036
Video-Transcoding	-11.059454228951008	-11.054351412935565	-16.06062412562616

With the exception of video-transcoding dataset, we see that overall, neural networks perform significantly better than linear regression both with and without regularization. There are several reasons for this:

- Neural networks are much more capable of capturing non-linear relationships in lieu of non-linear activations of weights compared to linear regression models.
- Neural networks have more information capacity (trainable parameters) than linear regression models, allowing MLP to capture complex mathematical relationships and mappings via hidden layer connections between features and target variables that is otherwise not possible to infer via linear dependencies with a small number of trainable weights.
- Neural networks automatically extract salient non-linear features from the input dataset without the need for explicit transformations, feature engineering or feature selection.

Find optimal hyperparameters (Question 18):

For this question, we test the performance in terms of train and test RMSE (average negative test RMSE, higher is better) of MLP for various weight decays, model activation functions and network size (number of hidden

neurons and depth) on bike-sharing, suicide-rates and video-transcoding datasets. Since F-Score selected features performed better than MI, we selected top 10 features from F-Score. The hyperparameter space was as follows:

- Model activation: {tanh, logistic, ReLU}
- Weight decay (for regularization): $[10^{-4}, 10^{2}]$
- Network sizes: Combinations from {10, 20, 30, 50} as the number of hidden neurons, with minimum network depth of 1 and maximum depth of 4.
  (10,), (20,), (30,), (50,), (10, 10), (10, 20), (10, 30), (10, 50), (20, 20), (20, 30), (20, 50), (30, 30), (30, 50), (50, 50), (10, 10, 10), (10, 10, 20), (10, 10, 30), (10, 10, 50), (10, 20, 20), (10, 20, 30), (10, 20, 50), (10, 30, 30), (10, 30, 50), (10, 50, 50), (20, 20, 20), (20, 20, 30), (20, 20, 50), (20, 30, 30), (20, 30, 50), (20, 50, 50), (30, 30, 30), (30, 30, 50), (30, 50, 50), (50, 50, 50), (10, 10, 10, 10), (10, 10, 10, 20), (10, 10, 10, 30), (10, 10, 10, 50), (10, 10, 20, 20), (10, 10, 20, 30), (10, 10, 20, 50), (10, 10, 30, 30), (10, 10, 30, 50), (10, 10, 50, 50), (10, 20, 20, 20), (10, 20, 20, 30), (10, 20, 20, 50), (10, 20, 30, 30), (10, 20, 30, 50), (10, 20, 50, 50), (10, 30, 30, 30), (10, 30, 30, 50), (10, 30, 50, 50), (10, 50, 50, 50), (20, 20, 20, 20), (20, 20, 20, 30), (20, 20, 20, 50), (20, 20, 30, 30), (20, 20, 30, 50), (20, 20, 50, 50), (20, 30, 30, 30), (20, 30, 30, 50), (20, 30, 50, 50), (20, 50, 50, 50), (30, 30, 30, 30), (30, 30, 30, 50), (30, 30, 50, 50), (30, 50, 50, 50), (50, 50, 50, 50)

We used 10-fold grid search cross-validation through the Sci-Kit Learn's `pipeline()` to find the most optimal hyperparameters for each dataset systematically in the parameter design space. Note that it is possible to use a much wider design space for network sizes, but to accelerate the search process, we limited the network sizes to choose combinations from four integers. Table 3 shows the optimal hyperparameter set, as well as the best average negative train and test RMSE obtained for each dataset.

Table 3. Performance summary of fully connected neural network with most optimal hyperparameters on bike-sharing, suicide-rates and video-trancoding datasets for top 10 features

Dataset (top 10 features from F-regression)	Best network Architecture	Best weight decay	Best activation function for hidden units	Train RMSE (negative, higher is better)	Test RMSE (negative, higher is better)
Bike-sharing	(10, 30, 30, 30)	10.0	ReLU	-763.5589843269157	-859.001551074912
Suicide-rates	(20, 30, 50, 50)	0.001	ReLU	-14.21613150345249	-14.10755903748036
Video-transcoding	(50, 50, 50, 50)	10.0	tanh	-16.02495236930304	-16.06062412562616

Choice of output activation function (Question 19):

Since the neural network is solving a regression problem with continuous outputs that can range from negative infinity to positive infinity, the output activation function should either be none or linear (identity), both of

which can provide outputs in the range $(-\infty, +\infty)$. ReLU activation is bounded to only 0 and positive values, tanh activation is bounded between -1 and 1 and sigmoid/logistic activation ranges from 0 to 1. However, regression outputs can take any value beyond such finite limits in the entire domain of real numbers, which is achievable via linear activation or no activation at all in the output layer.

Reasons for not having a very deep neural network (Question 20):

There are several reasons as to why we cannot (or should not) increase the depth of the neural network in an unconstrained manner:

- ***Overfitting:*** A deeper neural network has more trainable parameters than a shallow one, leading to a complex model with enough capacity to memorize the entire training set, consequently overfitting on the training data and generalizing poorly on the test set.
- ***Vanishing gradients:*** If a network is too deep, then the gradients will become exponentially small as the information back-propagates from the final layers towards the initial layers. This will lead to slow or premature convergence to a sub-optimal point during the training phase, with the weights in the initial layers being extremely small compared to the weights in the final layers.
- ***Compute constraints:*** A complex network with many layers takes more time, memory and compute power than a simpler model for training, neural architecture search and real-time deployment.
- ***Data requirements:*** If the network has millions of trainable weights, the neural network requires a large training set to avoid overfitting and provide robust and usable outputs during deployment.
- ***Interpretability:*** Large networks with millions of parameters can lead to complex relationships between features and target variables beyond human understanding, leading to black-box models.

Question 21 to 23:

In questions 21 to 23, we are asked to train a random forest ensemble, systematically explore its hyperparameter space and interpret the tree structure of one of them.

Finding optimal hyperparameters and their effects (Question 21):

For this question, we test the performance in terms of train and test RMSE (average negative test RMSE, higher is better) of random forests for various number of trees, maximum number of features and depth of each tree on bike-sharing, suicide-rates and video-transcoding datasets. Since F-Score selected features performed better than MI, we selected top 10 features from F-Score. The hyperparameter space was as follows:

- Maximum number of features: 1 to 10
- Number of trees/estimators: 10 to 200 (10 to 40 for video-transcoding dataset)
- Depth of each tree: 1 to 20

We used 10-fold grid search cross-validation through the Sci-Kit Learn's `pipeline()` to find the most optimal hyperparameters for each dataset systematically in the parameter design space. We also obtain the "Out-of-Bag Error" (OOB) for the best random forest models on each dataset (which we report in Question 28). To study the effects of each hyperparameter independently on the overall performance, for each of the three hyperparameters, we keep two of the hyperparameters constant while plotting the effects of the target hyperparameter on average train and test RMSE via 10-fold grid search cross-validation.

Table 4 shows the optimal hyperparameter set, as well as the best average negative train and test RMSE obtained for each dataset via random forest models.

Table 4. Performance summary of random forest models with most optimal hyperparameters on bike-sharing, suicide-rates and video-trancoding datasets for top 10 features

Dataset (top 10 features from F-regression)	Maximum number of features (best)	Depth of each tree (best)	Number of trees (best)	Train RMSE (negative, higher is better)	Test RMSE (negative, higher is better)
Bike-sharing	4	8	20	-276.8705434691136	-875.1063994497797
Suicide-rates	3	6	10	-14.21188672792134	-14.20312408548907
Video-transcoding	4	13	40	-0.746274747962649	-4.074124938241727

Figure 28, 29 and 30 show the effects of the number of trees on the overall model performance for bike-sharing dataset, suicide-rates dataset and video-transcoding dataset respectively.

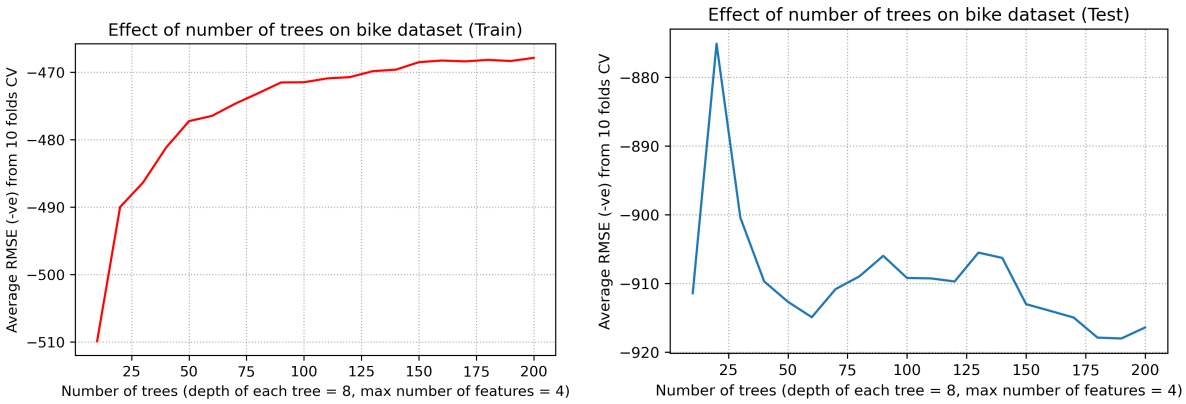


Figure 28: Effect of number of trees on bike sharing dataset (train and test).

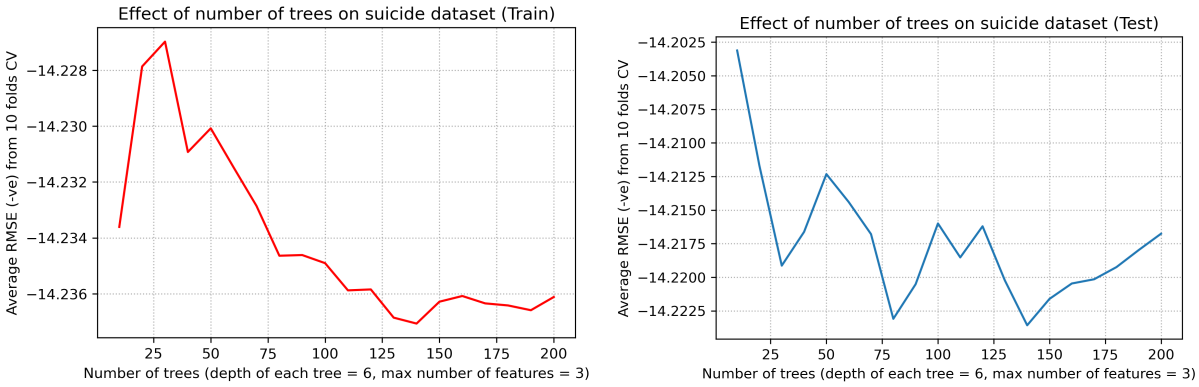


Figure 29: Effect of number of trees on suicide-rates dataset (train and test).

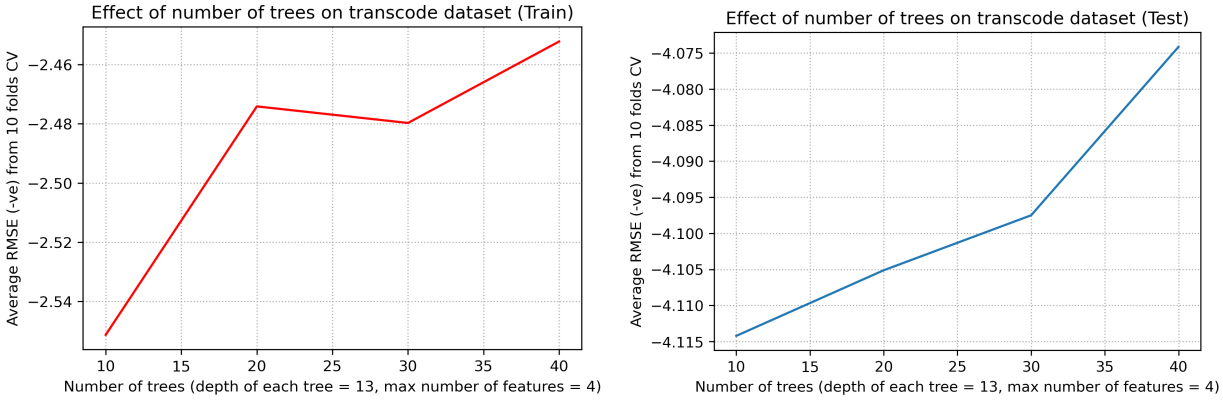


Figure 30: Effect of number of trees on video-transcoding dataset (train and test).

From Figures 28 to 30, overall, we see that the overall model performance is non-monotonous with respect to the number of trees, i.e. increasing the number of trees seem to improve or stabilize the performance but the performance improvement is non-monotonic. With all other hyperparameters fixed, the only effect the number

of trees have on the model's loss is to decrease it stochastically. One should select a sufficiently large value for the number of trees (e.g. 64 to 128) within compute constraints, as more number of trees does not cause overfitting.

- In a random forest, each tree along with its output target variable are both independent and identically distributed random variables (weak law of large numbers - WLLN) as the trees are grown using a randomization technique on their individual bootstrap subsamples uncorrelated with growth of other trees. Thus, according to WLLN, the target variable and tree-decision has finite variance, leading to the overall decision (and any other statistic) of the random forest to converge towards a mean value with diminishing returns when the number of trees approach infinity (Jensen's inequality).
- The expected error rate for a random forest ensemble is a non-monotonous function of the number of trees. In fact, error metrics such as RMSE are noisy once a sufficiently large number of required estimators have been used. The convergence rate of the error rate curve does not depend on the number of trees and is only dependent on the distribution of the expected value of the decisions of the trees.
- While increasing the number of trees does not cause overfitting according to WLLN, other hyperparameters might result in unnecessary variance in the model and cause over-correlation within the ensemble, breaking the notion of independent trees within the random forest.

Figure 31, 32 and 33 show the effects of the depth of each tree on the overall model performance for bike-sharing dataset, suicide-rates dataset and video-transcoding dataset respectively.

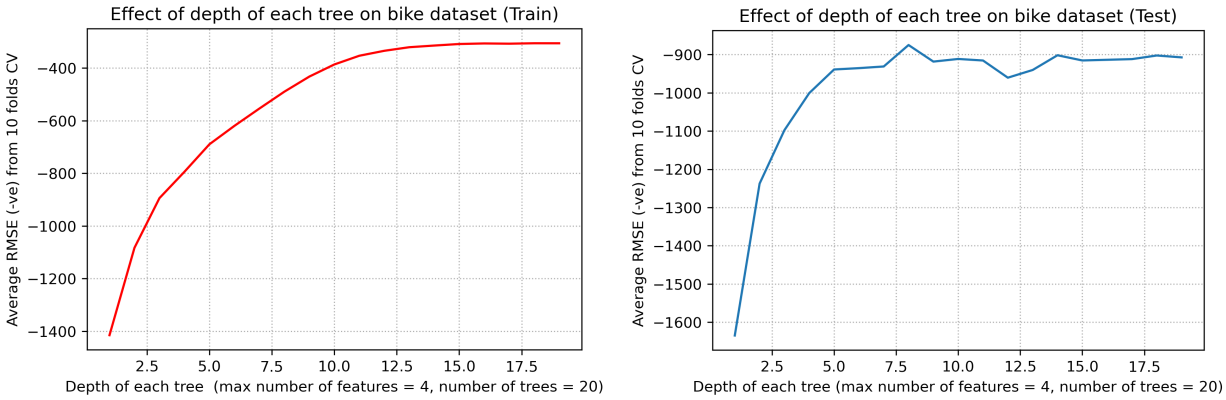


Figure 31: Effect of depth of each tree on bike sharing dataset (train and test).

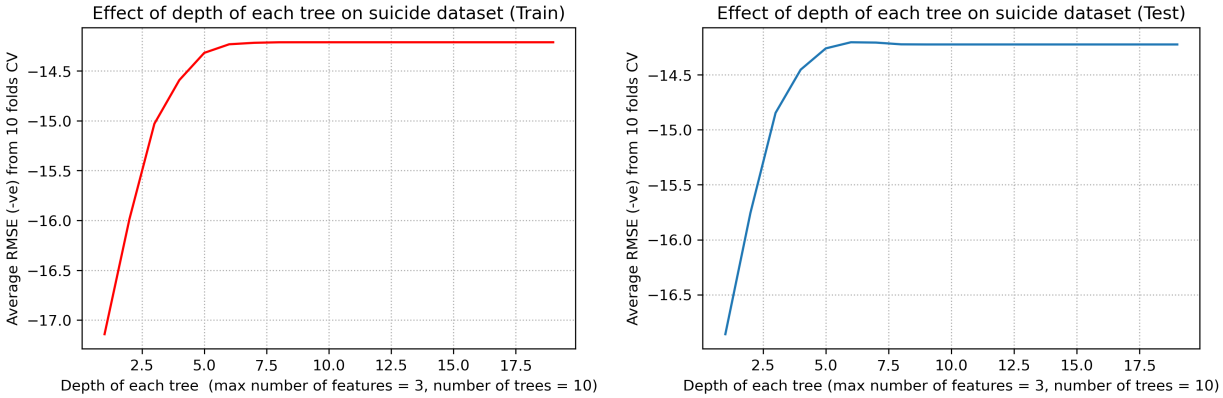


Figure 32: Effect of depth of each tree on suicide-rates dataset (train and test).

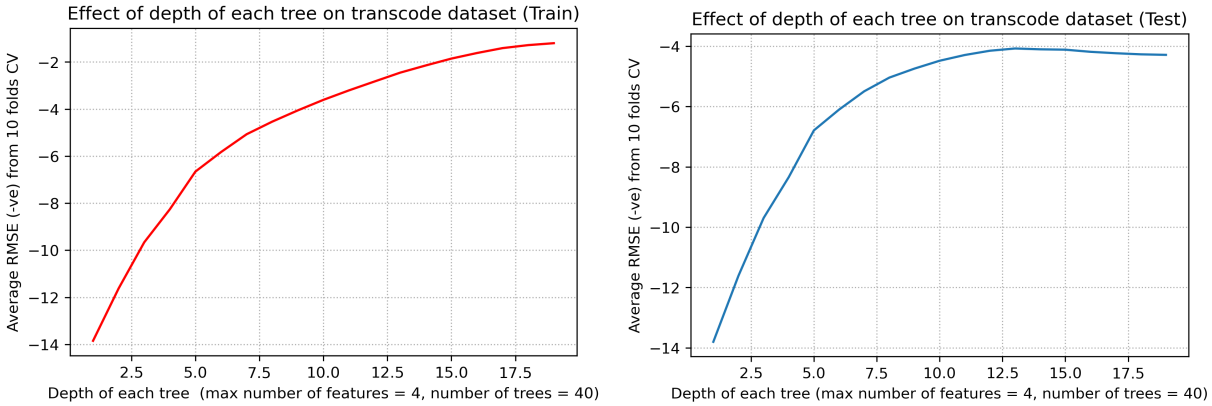


Figure 33: Effect of depth of each tree on video-transcoding dataset (train and test).

From Figures 31 to 33, for all three datasets, increasing the depth of each tree dramatically improves the error rate for both the training and test sets, converging to a constant value. For the test set, the error rate might start to degrade if the depth of each tree exceeds a certain threshold. This indicates that the depth of each tree acts as a regularizer, moderately increasing the depth of each tree improves both fitting and generalizability of the random forest, while very large values of depth of each tree will lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. A tree with a larger depth will have more fine-grained splits than a tree with shallower depth, having more model capacity and ability to fit complicated mathematical relationships, but also prone to overfitting. In other words, increasing tree depth decreases bias but increases variance.

Figure 34, 35 and 36 show the effects of the maximum number of features on the overall model performance for bike-sharing dataset, suicide-rates dataset and video-transcoding dataset respectively.

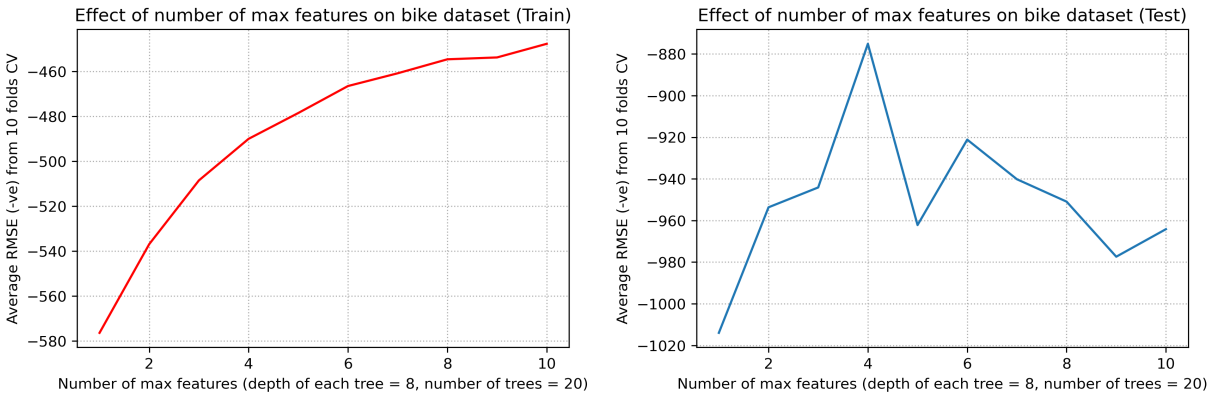


Figure 34: Effect of maximum number of features on bike sharing dataset (train and test).

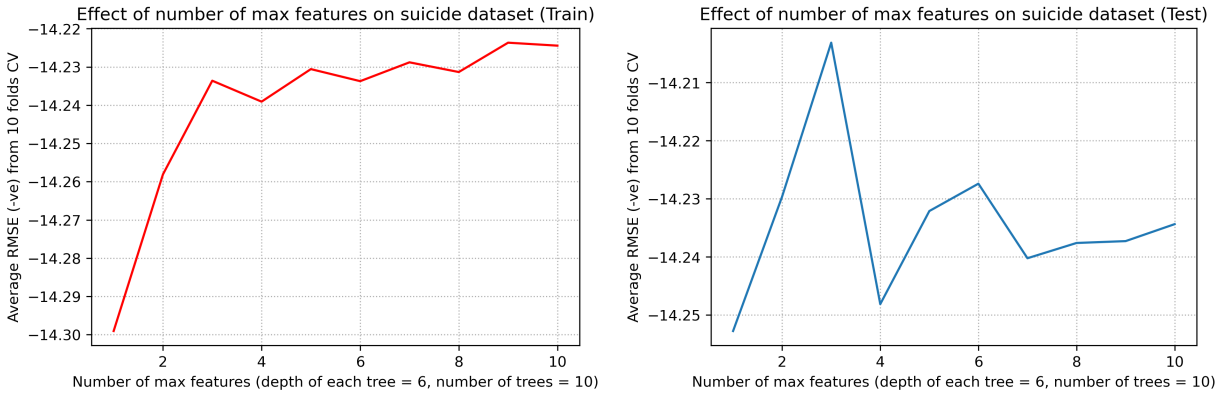


Figure 35: Effect of maximum number of features on suicide-rates dataset (train and test).

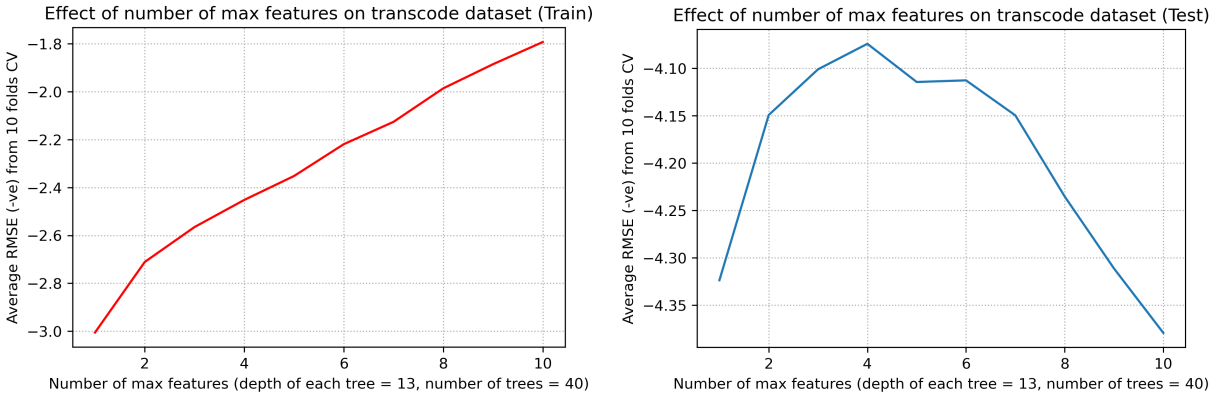


Figure 36: Effect of maximum number of features on video-transcoding dataset (train and test).

From Figures 34 to 36, for all three datasets, we observe that increasing the maximum number of features improves the training RMSE monotonically, converging to a mean value, while the test RMSE starts to degrade after showing improvement until a critical point. This indicates that the number of features also acts as a regularizer like depth of tree, with very large values leading to overfitting on the training set and poor generalization on the test set. This is because increasing the number of features for each tree improves the model capacity of individual trees, but also increases the correlation between each tree, breaking the notion of independent trees within the random forest. Selecting a fraction of the features aims to decorrelate the individual trees and improve generalization error rates, increasing the strength of the random forest.

Why random forest works well (Question 22):

In a random forest, each tree along with its output target variable are both independent and identically distributed random variables (weak law of large numbers - WLLN) as the trees are grown using a randomization technique on their individual bootstrap subsamples uncorrelated with growth of other trees. Thus, according to WLLN, the target variable and tree-decision has finite variance, leading to the overall decision (and any other statistic) of the random forest to converge towards a mean value with diminishing returns when the number of

trees approach infinity (Jensen's inequality). In other words, a random forest contains a large number of uncorrelated models, with the final target variable being a majority decision fusion from all of the trees. Bagging from independent models outperform individual models because of the following reasons:

- Mitigates errors or overfitting effects of individual models, assuming most of the trees are making correct decisions.
- Each decision tree is trained on a different and independent random subspace, forcing variation, decorrelation and diversification among individual trees. This ensures that all the features within the superspace are properly utilized by the random forest to form a decision.
- By ensembling, we are essentially getting the "mean of means", receiving a robust central estimate of the target variable from a collection of slightly different but overlapping weak learners, which combined, provides a robust model.
- Does not require scaling or standardization of features.

Visualizing decision trees (Question 23):

For this question, we trained a random forest of depth 4, with 3 maximum features and 10 trees on the suicide-rates dataset. We used `export_graphviz()` from Scikit-Learn and `pydot` library to visualize the one of the trees in that random forest model. The resulting tree is shown in Figure 37.

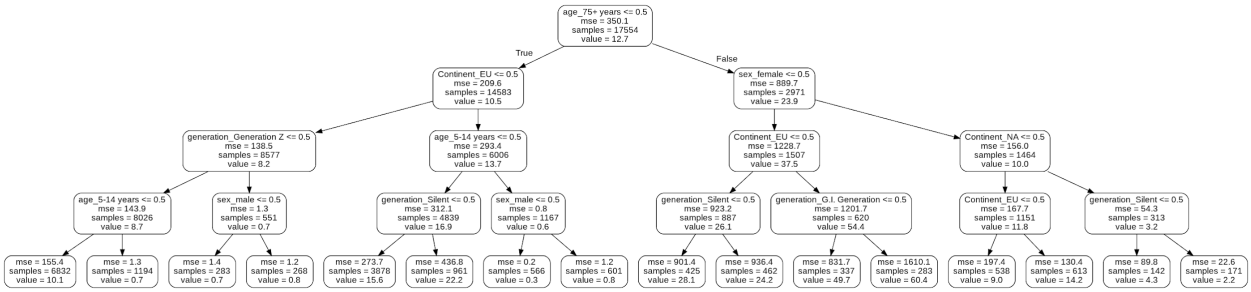


Figure 37: Sample tree from a random forest model trained on suicide-rates dataset.

From branching at the root node, 'age' is selected, more specifically, the dummy variable 'age_75+ years' is selected. This means that 'age_75+ years' is the most important feature used to start the splitting process. In a decision tree, the features closer to the root node are more salient and significant than the features near the leaf nodes. The top features (in order) for this sample tree are:

- Level 1: 'age_75+ years'
- Level 2: 'Continent_EU' and 'sex_female'
- Level 3: 'generation_Generation Z', 'age_5-14 years', 'Continent_EU' and 'Continent_NA'

Compare this with the top features we got from p-values in Question 13, which were gdp_per_capita ($), 'gdp_for_year ($)', 'age_75+ years', 'Continent_EU', 'age_55-74 years', 'sex_male', 'generation_G.I.

Generation' and 'age_35-54 years'. We do see 'age_75+ years' and 'Continent_EU' being the top features in both the cases, and overall, we see 'age', 'sex', 'generation' and 'continent' in both cases. Thus, it is safe to say that the most important features found from p-values of linear regression have significant overlap with the most important features found in a random decision tree within the random forest for the suicide-rates dataset.

Question 24 to 26:

In questions 24 to 26, we are asked to experiment with boosted trees, namely LightGBM and CatBoost, perform hyperparameter tuning on the named boosted tree methods using Bayesian optimization and visualize the effect of each of the individual hyperparameters on the trained models. For these questions, we chose to report results on the suicide-rates dataset, more specifically on the top 10 features selected via F-regression.

Most important hyperparameters of LightGBM and CatBoost (Question 24):

For lightGBM, we select the following hyperparameters along with associated proper search space:

- Boosting type ('boosting_type'): {gradient boosting decision tree (GBDT), Dropouts meet Multiple Additive Regression Trees (DART), Random Forest (RF)}. 'boosting_type' selects one of four gradient boosting algorithms (we omitted Gradient-based One-Side Sampling (GOSS) as it was throwing out errors during training):
    - GBDT (XGBoost): Default and most widely known boosted decision tree algorithm due to its accuracy, reliability and efficiency without requiring considerable effort for hyperparameter tuning. GBDT treats individual decision trees within an ensemble as weak learners, with the first tree aiming to fit the feature set to target variables and the succeeding trees aiming to reduce the residual error between the predicted target variable and ground truth target variable of preceding trees, with the entire ensemble trained via backpropagation of error gradients. The drawback of GBDT is the associated memory and compute requirements for finding the optimal split points for each individual tree, thereby not scaling well when the number of trees is very large. GBDT also suffers from over-specialization, with the latter trees making meaningful predictions for only a few samples in the dataset (sensitive to a few instances) and not generalizing well for the majority of the other samples
    - DART: DART solves the overspecialization problem in GBDT by introducing dropout, which is used in neural networks to drop weights for regularization and improving generalization for unseen test data.
    - RF: Random forest, which was introduced in Questions 21 to 23. In a random forest, each tree along with its output target variable are both independent and identically distributed random variables (weak law of large numbers - WLLN) as the trees are grown using a randomization technique on their individual bootstrap subsamples uncorrelated with growth of other trees. Thus, according to WLLN, the target variable and tree-decision has finite variance, leading to the overall decision (and any other statistic) of the random forest to converge towards a mean value with diminishing returns when the number of trees approach infinity (Jensen's inequality). In other words, a random forest contains a large number of uncorrelated models, with the final target variable being a majority decision fusion from all of the trees.
- Depth of each tree ('max_depth'): 1 to 90 in steps of 10. Increasing the depth of each tree improves the error rate for both the training and test sets, converging to a constant value. For the test set, the error

rate might start to degrade if the depth of each tree exceeds a certain threshold. This particular hyperparameter acts as a regularizer and performance contributor, moderately increasing the depth of each tree improves both fitting and generalizability, while very large values of depth of each tree will lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. A tree with a larger depth will have more fine-grained splits than a tree with shallower depth, having more model capacity and ability to fit complicated mathematical relationships, but also prone to overfitting and large training time. In other words, increasing tree depth decreases bias but increases variance.

- Number of leaves ('num_leaves'): 20 to 990 in steps of 10. Similar to the depth of each tree, the number of leaf nodes acts as a regularizer and performance contributor, moderate values improve both fitting and generalizability, while very large values lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. LightGBM adds leaf nodes to trees based on performance gains from adding those leaves, regardless of depth of each tree. As a result, it is necessary to tune both the maximum number of leaves as well as the depth of each tree to control model complexity.
- Number of trees ('n_estimators'): 10 to 3900 in steps of 100. Specifies the number of boosted trees to fit. Increasing the number of trees improves and stabilizes model performance non-monotonically. The expected error rate for a boosted tree ensemble is a non-monotonous function of the number of trees, being noisy once a sufficiently large number of required estimators have been used. With all other hyperparameters fixed, the only effect the number of trees have on the model's loss is to decrease it stochastically. One should select a sufficiently large value for the number of trees within compute constraints, as more number of trees does not cause overfitting similar to random forests. The overall decision of the ensemble converges towards a mean value with diminishing returns when the number of trees approach infinity (Jensen's inequality). Too few trees, however, will hurt model performance.
- L1 regularization (alpha) penalty parameter ('reg_alpha'): $[10^{-4}, 10^{4}]$. Lasso regularization penalty term on the weights. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization but may hurt model performance.
- L2 regularization (lambda) penalty parameter ('reg_lambda'): $[10^{-4}, 10^{4}]$. Tikhonov regularization penalty term on the weights. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization but may hurt model performance.
- Subsampling ratio or bagging fraction ('subsample'): $[0.1, 1]$. Specifies the fraction of rows (samples) to be randomly sampled for fitting each tree. The process of training over multiple random samples without replacement is called "bagging", and is similar to how individual trees in a random forest are fitted. Decreasing subsampling ratio helps with overfitting and generalization issues and acts as a regularizer; very small values of bagging fraction may hurt model performance while very large values (no bagging) improves model performance on the training set but generalizes poorly on the test set. Bagging aims at reducing the variance of the ensemble. A similar hyperparameter for selecting features is feature fraction ('colsample_bytree') or random subspace method, which we do not use as we already selected the 10 most salient features for the ensemble to work on. We do however, test this on CatBoost for sake of completeness.

- Subsampling frequency or bagging frequency ('subsample_freq'): 0 to 45 in steps of 5. Controls how often bagging is performed, in terms of epochs.
- Minimal gain to perform split ('min_split_gain'): $[10^{-4}, 10^{0}]$. Minimum reduction in training loss that results from adding further split points required to partition a leaf node of the tree. LightGBM selects the split point that has the highest gain when adding a new tree node. Increasing this ratio acts as a regularizer, causing the ensemble to ignore very small training performance improvements that do not have meaningful impacts on generalizability of the model.

For CatBoost, we select the following hyperparameters along with associated proper search space:

- Feature fraction ('colsample_bylevel' or 'rsm'): $$[0.1, 1]$$. This is a random subspace method that selects the fraction of features to use at each split. Decreasing feature fraction helps with overfitting and generalization issues by selecting only the most salient and important features that have an impact on most of the samples, acting as a regularizer; very small values of feature fraction may hurt model performance while very large values (no random subspace) improves model performance on the training set but generalizes poorly on the test set. Random subspace aims at reducing the variance of the ensemble.
- Number of trees ('num_trees' or 'num_boost_round' or 'n_estimators'): 10 to 1000 in steps of 100. Specifies the number of boosted trees to fit. Increasing the number of trees improves and stabilizes model performance non-monotonically. The expected error rate for a boosted tree ensemble is a non-monotonous function of the number of trees, being noisy once a sufficiently large number of required estimators have been used. With all other hyperparameters fixed, the only effect the number of trees have on the model's loss is to decrease it stochastically. One should select a sufficiently large value for the number of trees within compute constraints, as more number of trees does not cause overfitting similar to random forests. The overall decision of the ensemble converges towards a mean value with diminishing returns when the number of trees approach infinity (Jensen's inequality). Too few trees, however, will hurt model performance.
- Number of leaves ('num_leaves' or 'max_leaves'): 20 to 990 in steps of 10. This is used only for lossguide growing policy. Similar to the depth of each tree, moderate values improve both fitting and generalizability, while very large values lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree.
- L2 regularization penalty parameter ('l2_leaf_reg' or 'reg_lambda'): $$[10^{-4}, 10^4]$$. Tikhonov regularization penalty term on the weights. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization but may hurt model performance
- Depth of each tree ('max_depth' or 'depth'): 1 to 16 (16 is the maximum CatBoost supports on CPU) in steps of 2. Increasing the depth of each tree improves the error rate for both the training and test sets, converging to a constant value. For the test set, the error rate might start to degrade if the depth of each tree exceeds a certain threshold. This particular hyperparameter acts as a regularizer and performance contributor, moderately increasing the depth of each tree improves both fitting and generalizability,

while very large values of depth of each tree will lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. A tree with a larger depth will have more fine-grained splits than a tree with shallower depth, having more model capacity and ability to fit complicated mathematical relationships, but also prone to overfitting and large training time. In other words, increasing tree depth decreases bias but increases variance.

- Bagging temperature ('bagging_temperature'): $$[0.1, 10]$$. CatBoost uses Bayesian bootstrap aggregation for regression problems. If bagging temperature is 0, the weights assigned to the sampled subspaces are equal to 1. If bagging temperature increases, the intensity of bootstrap increases and the weights are sampled from exponential distribution. CatBoost uses minimum variance sampling or weighted sampling at the tree level and not the split level.
- Grow policy ('grow_policy'): {Symmetric Tree, Depthwise Tree and LossGuide}. Specifies how the trees will be generated from the leaves. Unlike LightGBM which uses leaf-wise tree growth (best-first) to allow for an imbalance tree (may cause overfitting for small datasets) regardless of the depth of each tree, Catboost grows a balanced tree such that at each level, the feature-split pair that minimizes the loss function is selected and utilized for all level nodes.
    - Symmetric Tree (ST): Level by level growth strategy; on each iteration, all leaves from the last tree level are split with the same condition.
    - Depthwise Tree (DT): Level by level growth strategy; on each iteration, all non-terminal leaves from the last tree level are split depending on the feature-split pair that minimizes the loss function.
    - LossGuide (similar to LightGBM) (LG): Leaf by leaf growth strategy; on each iteration, non-terminal leaf with the best loss improvement is split.
- Score function ('score_function'): {Cosine, L2}. This is used only for symmetric or depthwise growing policies. Score function is used to choose between candidate trees when adding a new tree to the ensemble. L2 and Cosine are first-order score functions.

Hyperparameter tuning with Bayesian optimization (Question 25):

For this question, we find the optimal hyperparameters in terms of train and test RMSE (average negative test RMSE, higher is better) of boosted trees (LightGBM and CatBoost) using Bayesian optimization on the chosen dataset (suicide-rates dataset). In Bayesian optimization, a Gaussian process is used to approximate the objective function (called surrogate model), which allows priors on the distribution of moments (mean and uncertainty) to propagate forward as the search progresses. The acquisition function decides the next set of hyperparameters to sample from the design space using Monte Carlo sampling with Bayesian Upper-Confidence Bounds (UCB), also known Thompson sampling, which balances exploration and exploitation. Thompson sampling ensures that the acquisition function does not get stuck in a local optimum early in the search, with the exploration parameter decreased as the confidence in the Pareto-frontier grows. We used 10 iterations for the CatBoost regressor, and 20 iterations for the LightGBM regressor, coupled with 10-fold cross-validation. Table 5 outlines the best set of hyperparameters obtained for both the regressors. From Table 5, we see that lightGBM performs

slightly better than CatBoost in terms of average train and test RMSE on the suicide-rates dataset. However, there is no significant performance difference among the two classifiers. LightGBM is more robust to overfitting than catboost, possibly due to the presence of more regularization hyperparameters than CatBoost.

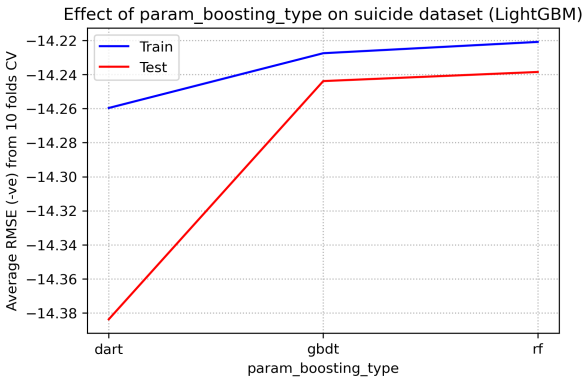
Table 5: Optimal hyperparameters and corresponding average negative train and test RMSE found using Bayesian optimization with 10-fold cross-validation on suicide-rates dataset

Regressor	Boosting Type	Depth of each tree	Number of trees	Number of leaves	Min. split gain	Bagging fraction	Bagging frequency	L1 reg.	L2 reg.	Feature fraction	Grow policy, score func.	Bagging temp.	Train RMSE (-ve)	Test RMSE (-ve)
LightGBM	GBDT	81	710	800	0.0001	0.5	35	0.1	0.1	default	-	-	-14.28	-14.208
CatBoost	-	9	210	N/A for ST	-	-	-		0.01	0.8	ST, Cosine	0.1	-17.24	-14.22

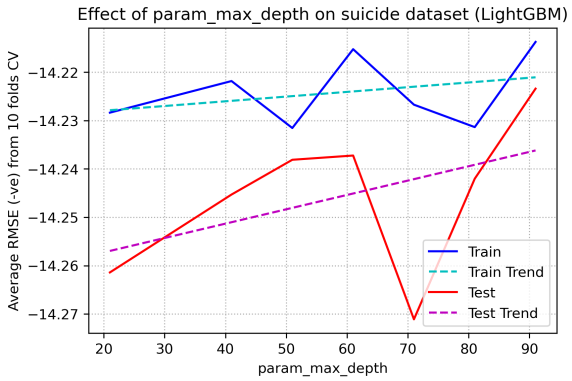
Effect of hyperparameters (Question 26):

Figure 38 shows the effects of the 9 hyperparameters of LightGBM on the average negative train and test RMSE for suicide-rates dataset obtained from Bayesian hyperparameter tuning. Since Bayesian optimization is stochastic, we also plot trend lines to showcase the overall tendency of each hyperparameter.

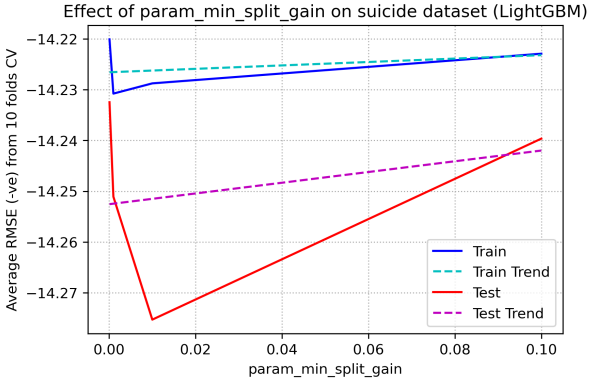
1



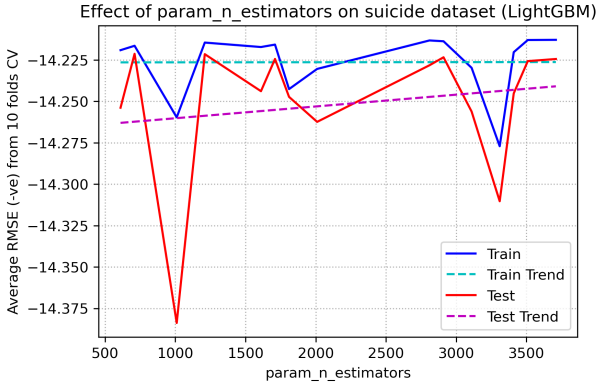
2



3



4



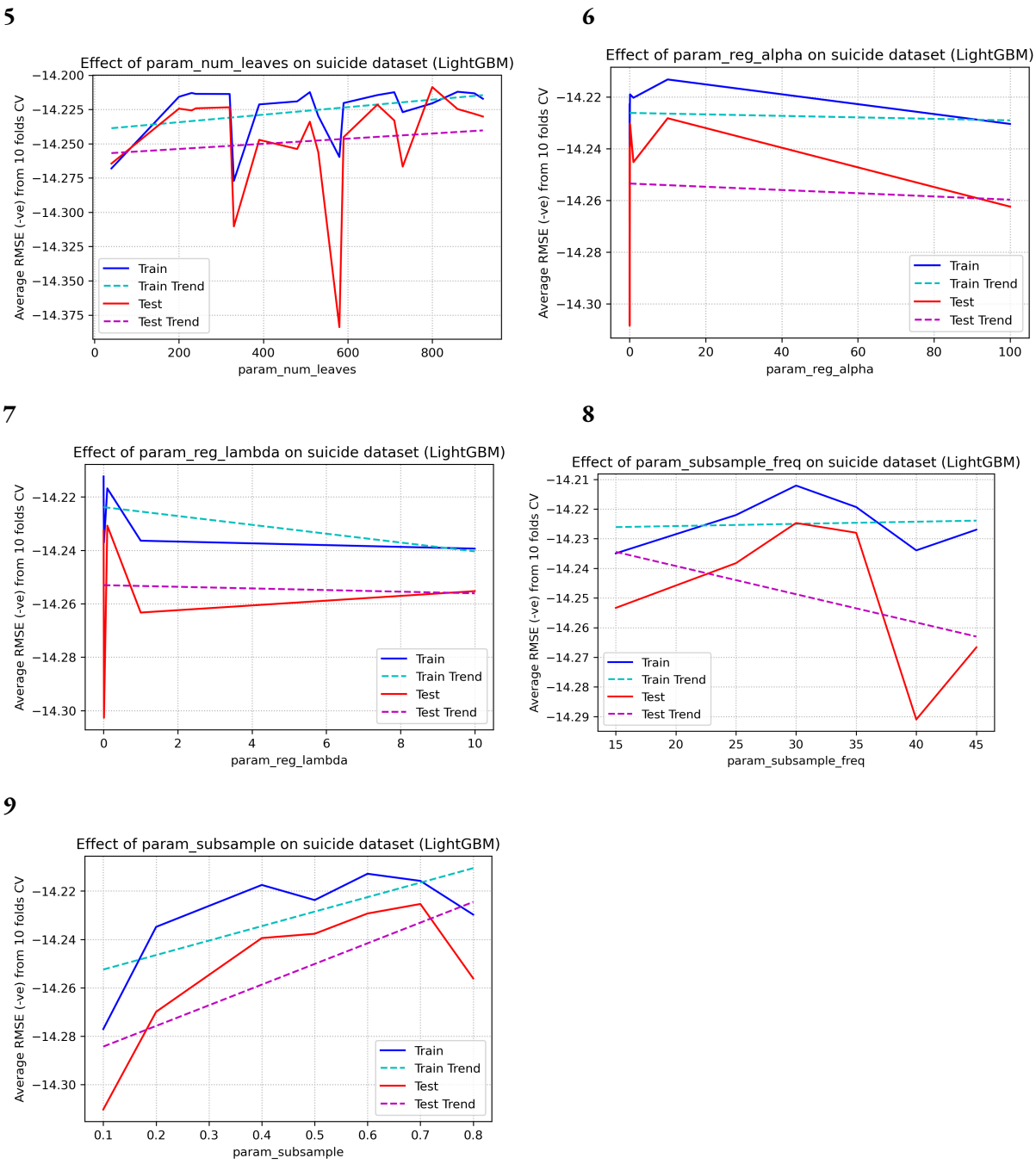


Figure 38: Effect of the 9 hyperparameters of LightGBM individually on suicide-rates dataset.

1. Boosting Type: From Figure 38 (1), we see that GBDT and RF perform similar to each other, while both jointly outperform DART. Theoretically, DART should improve generalizability of GBDT by introducing dropout. However, DART also has implicit hyperparameters that require tuning, such as model seed, the probability of skipping dropout during an iteration, dropout rate and whether to use XGBoost dropout or uniform dropout, which we did not tune due to compute constraints. Too many floating parameters can result in unexpected performance of DART.

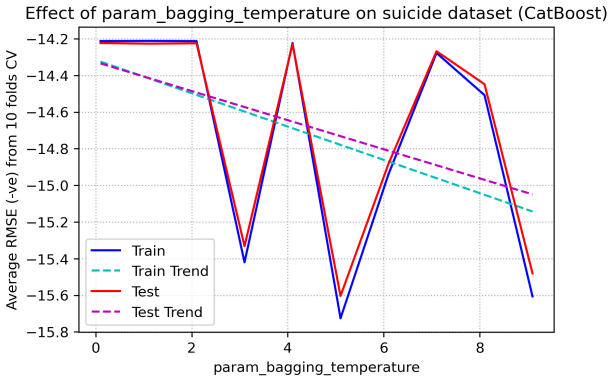
2. Depth of each tree: From Figure 38 (2), we see that increasing the depth of each tree improves the error rate for both the training and test sets slightly. This particular hyperparameter acts both as a regularizer and a performance contributor, moderately increasing the depth of each tree improves both fitting and generalizability, while very large values of depth of each tree will lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. A tree with a larger depth will have more fine-grained splits than a tree with shallower depth, having more model capacity and ability to fit complicated mathematical relationships, but also prone to overfitting and large training time. In other words, increasing tree depth decreases bias but increases variance.
3. Minimal gain to perform split: From Figure 38 (3), we see that the both training and test RMSE improves slightly with an increase in the gain ratio. Increasing this ratio acts as a regularizer, causing the ensemble to ignore very small training performance improvements that do not have meaningful impacts on generalizability of the model. However, the graph does not show the full picture as the Bayesian agent did not cover the full range of the ratio. More aggressive bagging is likely to cause drop on model performance due to regularization effects.
4. Number of trees: From Figure 38 (4), we do not see any performance gains or losses as the number of trees increases. Increasing the number of trees may improve and stabilize model performance non-monotonically. The expected error rate for a boosted tree ensemble is a non-monotonous function of the number of trees, being noisy once a sufficiently large number of required estimators have been used. With all other hyperparameters fixed, the only effect the number of trees have on the model's loss is to decrease it stochastically. One should select a sufficiently large value for the number of trees within compute constraints, as more number of trees does not cause overfitting similar to random forests. The overall decision of the ensemble converges towards a mean value with diminishing returns when the number of trees approach infinity (Jensen's inequality). Too few trees, however, will hurt model performance (weak law of large numbers). Thus, unless the number of trees is very small, increasing the number of trees has no effect on the model performance.
5. Number of leaves ('num_leaves' or 'max_leaves'): Similar to the depth of each tree, the number of leaf nodes acts as a regularizer and performance contributor, moderate values improve both fitting and generalizability, while very large values lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. Figure 38 (5) shows slight improvement in performance as the number of leaves increases.
6. L1 regularization term: From Figure 38 (6), we observe that both train and test RMSE initially improves with finite values of Lasso regularization, but performance drops when aggressive regularization is performed. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization but may hurt model performance.
7. L2 regularization term: From Figure 38 (7), we observe that both train and test RMSE initially improves with finite values of Tikhonov regularization, but performance drops when aggressive regularization is performed. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization but may hurt model performance. Note that L1 regularization is a stronger regularization and causes more aggressive regularization than

L2, as L1 regularization zeros out weights more often than L2 regularization. From more details on L1 and L2 regularization, we urge the grader to look at Question 11.

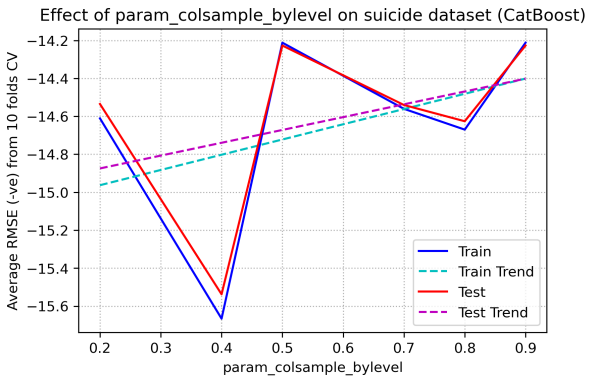
8. Bagging frequency: From Figure 38 (8), we see that as bagging is conducted more often, the performance drops. This is expected because bagging acts as a regularizer. More frequent bagging (dropping samples) drops a subset of training data more frequently, which can cause important training samples to never be considered during training.
9. Bagging fraction: From Figure 38 (9), we see that as subsampling ratio increases, the performance also increases monotonically. Decreasing subsampling ratio helps with overfitting and generalization issues and acts as a regularizer; very small values of bagging fraction may hurt model performance while very large values (no bagging) improves model performance on the training set but generalizes poorly on the test set. Bagging aims at reducing the variance of the ensemble.

Figure 39 shows the effects of the 8 hyperparameters of CatBoost on the average negative train and test RMSE for suicide-rates dataset obtained from Bayesian hyperparameter tuning. Since Bayesian optimization is stochastic, we also plot trend lines to showcase the overall tendency of each hyperparameter.

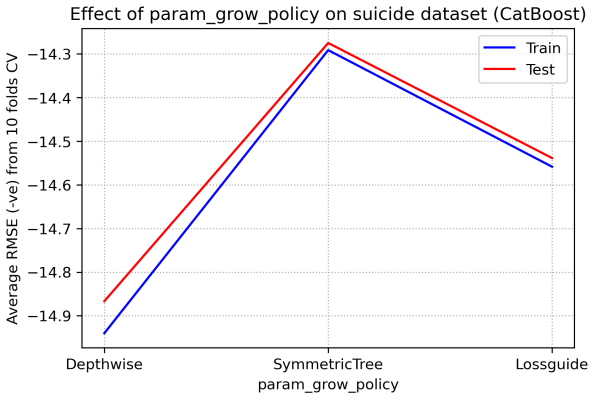
1



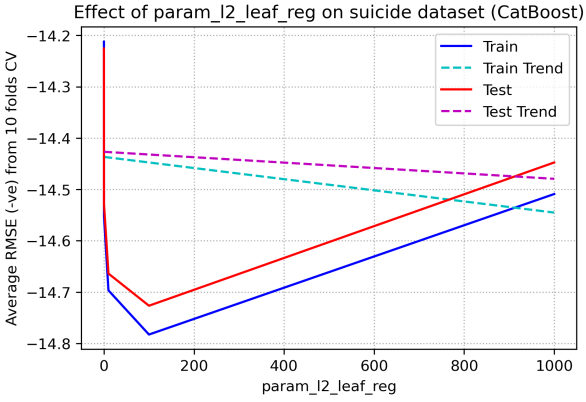
2



3



4



5

6

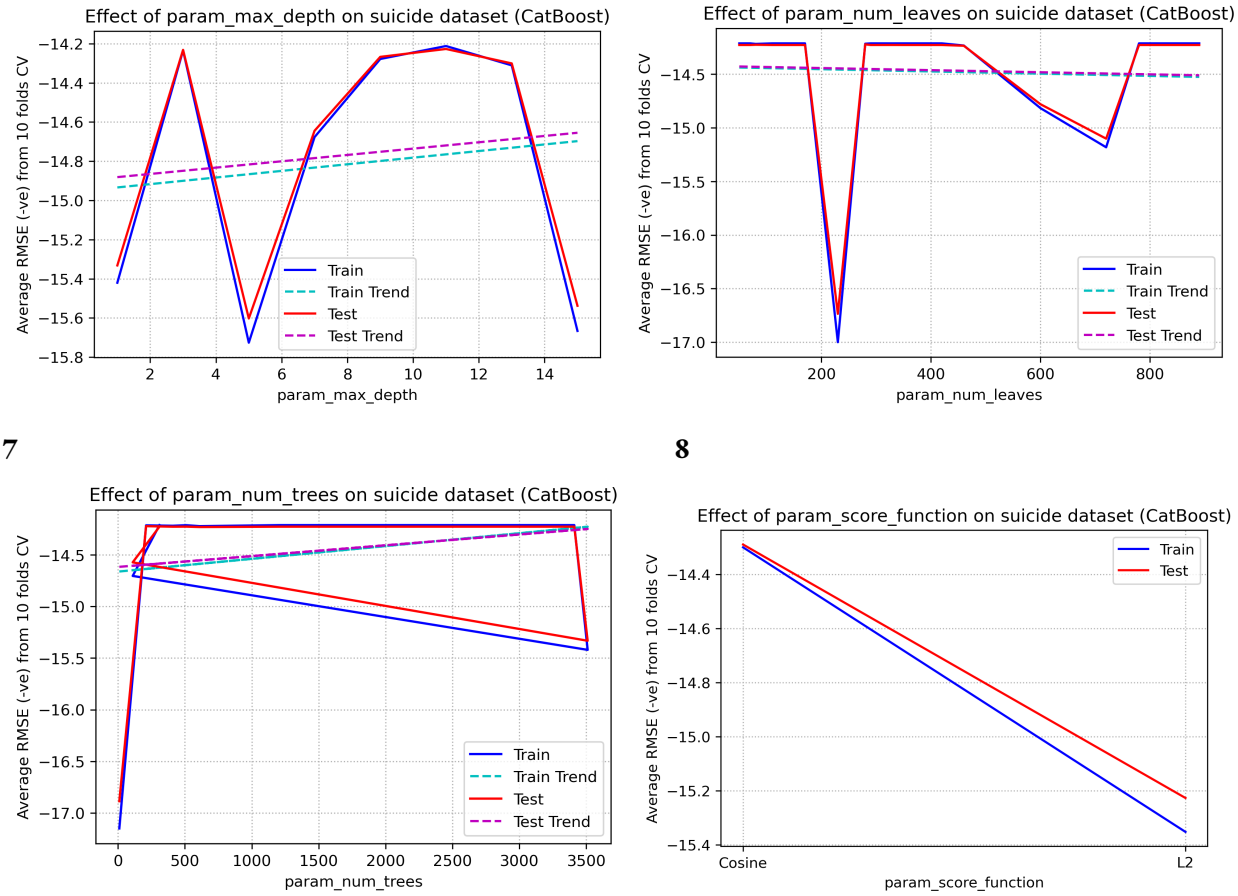


Figure 39: Effect of the 8 hyperparameters of CatBoost individually on suicide-rates dataset.

1. Bagging temperature: From Figure 39(1), we see that as bagging temperature increases, the tran and test RMSE degrades. If bagging temperature is 0, the weights assigned to the sampled subspaces are equal to 1. If bagging temperature increases, the intensity of bootstrap increases and the weights are sampled from exponential distribution. Since, we are already supplying top 10 salient features to CatBoost, assigning extreme weights to these features can hurt model performance.
2. Feature fraction: From Figure 39 (2), we see that as feature fraction increases, the performance improves as well. Decreasing feature fraction helps with overfitting and generalization issues by selecting only the most salient and important features that have an impact on most of the samples, acting as a regularizer; very small values of feature fraction may hurt model performance while very large values (no random subspace) improves model performance on the training set but generalizes poorly on the test set. Random subspace aims at reducing the variance of the ensemble.
3. Grow policy: From Figure 39 (3), we observe that symmetric growth policy performs better than both depthwise and lossguide growth policies. Lossguide uses leaf-wise tree growth (best-first) to allow for an imbalance tree, which usually causes overfitting when the feature space is small. In addition, symmetric tree considers all leaves from the last iteration instead of just the non-terminal leaves for making splits, thus having a more generalized overview of the earlier feature subspaces than depthwise tree.

4. L2 regularization: From Figure 39 (4), we observe that overall, more aggressive regularization leads to performance drops. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization but may hurt model performance.
5. Depth of each tree: From Figure 39 (5), we see that increasing the depth of each tree improves the error rate for both the training and test sets slightly. This particular hyperparameter acts both as a regularizer and a performance contributor, moderately increasing the depth of each tree improves both fitting and generalizability, while very large values of depth of each tree will lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. A tree with a larger depth will have more fine-grained splits than a tree with shallower depth, having more model capacity and ability to fit complicated mathematical relationships, but also prone to overfitting and large training time. In other words, increasing tree depth decreases bias but increases variance.
6. Number of leaves (only for lossguide): Similar to the depth of each tree, the number of leaf nodes acts as a regularizer and performance contributor, moderate values improve both fitting and generalizability, while very large values lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. Figure 39 (6), however, shows no significant performance changes with changes in number of leaves, which is probably because other hyperparameters rendered the leaf hyperparameter change contribute insignificantly towards the final performance.
7. Number of trees: From Figure 39 (7), we see slight performance gains as the number of trees increases. Increasing the number of trees may improve and stabilize model performance non-monotonically. The expected error rate for a boosted tree ensemble is a non-monotonous function of the number of trees, being noisy once a sufficiently large number of required estimators have been used. With all other hyperparameters fixed, the only effect the number of trees have on the model's loss is to decrease it stochastically. One should select a sufficiently large value for the number of trees within compute constraints, as more number of trees does not cause overfitting similar to random forests. The overall decision of the ensemble converges towards a mean value with diminishing returns when the number of trees approach infinity (Jensen's inequality). Too few trees, however, will hurt model performance (weak law of large numbers). Thus, unless the number of trees is very small, increasing the number of trees has no effect on the model performance.
8. Score function: From Figure 39 (8), we see that cosine distance performs better as a score function than L2 distance. This is expected because cosine similarity is not affected by the magnitude of the features, meaning that the range and scale of the features does not affect the distance metric, but rather it associates features based on angle between sample points. This corrects the effects of non-standardized or unscaled features. In addition, in high dimensions, the rapid increase in volume causes the data to become sparse in each dimension, causing the Euclidean distances to converge to a constant value between all sample points. Since all feature points become equidistant, models working with L2 distances cannot find distinguishable features within the data. Thus, for textual clustering, cosine similarity is the ideal metric over L2 distances

Question 27:

In this question, we are asked to perform 10-fold cross-validation and measure average RMSE errors for training and validation sets for all models and all three datasets. We already performed this task earlier and urge the grader to look at Questions 10 to 26 for more detailed analysis. Here, we report the performance of the best model (hyperparameter tuned either using 10-fold grid search cross validation or Bayesian optimization) found via 10-fold grid-search cross validation for each dataset in Table 6. For the optimal list of hyperparameters, please consult Questions 10 to 26. The best model for bike-sharing dataset was polynomial regression, multilayer perceptron for suicide-rates and random forest for video-transcoding, with top 3 models in each set numbered in braces.

Table 6: Best-performance of each model on bike-sharing, suicide-rates and video-transcoding dataset

Model	**Best average train RMSE (negative, higher is better)**			**Best average test RMSE (negative, higher is better)**		
	Bike-Sharing	**Suicide-Rates**	**Video-Transcoding**	**Bike-Sharing**	**Suicide-Rates**	**Video-Transcoding**
Linear Regression	-836.3621038950808	-15.420589566680139	-11.038710923489962	-904.875354514855	-15.332744465450682	-11.059454228951008
Linear Regression (Lasso)	-836.3621040459991	-15.420182153255216	-11.038711188883545	-897.4951623793668	-15.327509793376663	-11.054351412935565 (3)
Linear Regression (Ridge)	-836.3621038954203	-15.420182151332 71	-11.038710923489962	-878.1469958606837	-15.322322304708754	-11.059033334149994
Polynomial Regression	-550	-14.20	-6.5	-851 (1)	-14.22	-6.8 (2)
Multilayer Perceptron	-763.5589843269157	-14.216131503452 49	-16.024952369303 04	-859.001551074912 (2)	-14.107559037480 36 (1)	-16.060624125626 16
Random Forest	-276.8705434691136	-14.211886727921 34	-0.7462747479626 49	-875.106399449779 7 (3)	-14.203124085489 07 (2)	-4.0741249382417 27 (1)
LightGBM	-	-14.28	-	-	-14.208 (3)	-
CatBoost	-	-17.24	-	-	-14.22	-

We see that usually, the negative training RMSE is higher (**absolute training RMSE is lower**) than the test (validation) RMSE. This happens because complex models tend to overfit on the characteristics of the training set, which may not be present in the validation set and lead to models not generalizing well on the validation set.

Question 28:

In this question, we report the OOB error for the random forest models in Questions 21 to 23. The OOB error for the best random forest models on the three datasets are as follows:

- OOB, bike-sharing: 0.8274957524625527
- OOB, suicide-rates: 0.4220842787293496
- OOB, video-transcoding: 0.9601340475580523

In a random forest, each tree along with its output target variable are both independent and identically distributed random variables (weak law of large numbers - WLLN) as the trees are grown using a randomization technique on their individual bootstrap subsamples uncorrelated with growth of other trees. Each decision tree is trained on a different and independent random subspace, forcing variation, decorrelation and diversification among individual trees. OOB score is a technique for validating random forest models, defined as the number of correctly predicted samples (rows) from the out of bag sample rows via majority decision fusion. An out-of-bag sample is defined rows in the original dataset that was not present in the bootstrap samples of a particular decision tree. Since OOB is being calculated on unseen test data, it serves as a form of validation score for the ensemble.

R^2 (coefficient of determination), on the other hand, is defined as the ratio of the variance in the target variable that can be predicted by the features in the dataset (also known as goodness of fit). It includes all the data in the training (and validation) set regardless of whether a tree within an ensemble saw some of the samples or not, along with decision from all the decision trees. For R^2 score calculation, separate steps are required for training and validation score calculation.

project_219_4_Saha_Young

March 12, 2021

[4]:
```
import pandas as pd
pd.set_option('display.max_columns', None)
import matplotlib.pyplot as plt
import numpy as np
from pandas_profiling import ProfileReport
import seaborn
import pycountry_convert as pc
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.feature_selection import SelectKBest, mutual_info_regression,␣
↪f_regression
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.model_selection import cross_validate, GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.neural_network import MLPRegressor
from statsmodels.regression.linear_model import OLS
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import export_graphviz
import pydot
from statsmodels.api import add_constant
import itertools
from IPython.display import Image
from skopt import BayesSearchCV
from catboost import CatBoostRegressor
import lightgbm as lgb
```

0.1 Question 1, 2 and 6

[5]:
```
dataset_folders = ['Bike-Sharing-Dataset/', 'Suicide_Rates/
↪','online_video_dataset/']
```

0.1.1 Bike Sharing Dataset

[6]:
```
bike = pd.read_csv(dataset_folders[0]+"day.csv")
bike = bike.drop(['instant'],axis=1)
print(bike)
bike_prof = ProfileReport(bike, title="Profile Report for Bike Sharing Dataset")
bike_prof.to_widgets()
```

```
bike_prof.to_file('bike_prof.html')
```

```
         dteday  season  yr  mnth  holiday  weekday  workingday  weathersit  \
0    2011-01-01       1   0     1        0        6           0           2
1    2011-01-02       1   0     1        0        0           0           2
2    2011-01-03       1   0     1        0        1           1           1
3    2011-01-04       1   0     1        0        2           1           1
4    2011-01-05       1   0     1        0        3           1           1
..          …     …   ..    …       …       …          …         …
726  2012-12-27       1   1    12        0        4           1           2
727  2012-12-28       1   1    12        0        5           1           2
728  2012-12-29       1   1    12        0        6           0           2
729  2012-12-30       1   1    12        0        0           0           1
730  2012-12-31       1   1    12        0        1           1           2

         temp     atemp       hum  windspeed  casual  registered   cnt
0    0.344167  0.363625  0.805833   0.160446     331         654   985
1    0.363478  0.353739  0.696087   0.248539     131         670   801
2    0.196364  0.189405  0.437273   0.248309     120        1229  1349
3    0.200000  0.212122  0.590435   0.160296     108        1454  1562
4    0.226957  0.229270  0.436957   0.186900      82        1518  1600
..        …       …         …          …     …          …   …
726  0.254167  0.226642  0.652917   0.350133     247        1867  2114
727  0.253333  0.255046  0.590000   0.155471     644        2451  3095
728  0.253333  0.242400  0.752917   0.124383     159        1182  1341
729  0.255833  0.231700  0.483333   0.350754     364        1432  1796
730  0.215833  0.223487  0.577500   0.154846     439        2290  2729

[731 rows x 15 columns]
```

```
Summarize dataset:   0%|          | 0/28 [00:00<?, ?it/s]
```

```
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
```

```
Render widgets:   0%|          | 0/1 [00:00<?, ?it/s]
```

```
VBox(children=(Tab(children=(Tab(children=(GridBox(children=(VBox(children=(GridspecLayout(chil
```

```
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

0.1.2 Suicide Rates Dataset

[7]:
```
cols_to_use = ['country','year','sex','age','population',' gdp_for_year ($)␣
↪','gdp_per_capita ($)','generation',

                                                                    ␣
↪'suicides_no','suicides/100k pop']
suicide = pd.read_csv(dataset_folders[1]+"master.
↪csv",thousands=',',usecols=cols_to_use)[cols_to_use]
```

```
print(suicide)
suicide_prof = ProfileReport(suicide, title="Profile Report for Suicide Rates␣
 ↪Dataset")
suicide_prof.to_widgets()
suicide_prof.to_file('suicide_prof.html')
```

```
          country  year     sex          age  population   gdp_for_year ($)   \
0         Albania  1987    male  15-24 years      312900          2156624900
1         Albania  1987    male  35-54 years      308000          2156624900
2         Albania  1987  female  15-24 years      289700          2156624900
3         Albania  1987    male    75+ years       21800          2156624900
4         Albania  1987    male  25-34 years      274300          2156624900
…           …    …     …          …          …                  …
27815  Uzbekistan  2014  female  35-54 years     3620833         63067077179
27816  Uzbekistan  2014  female    75+ years      348465         63067077179
27817  Uzbekistan  2014    male   5-14 years     2762158         63067077179
27818  Uzbekistan  2014  female   5-14 years     2631600         63067077179
27819  Uzbekistan  2014  female  55-74 years     1438935         63067077179

       gdp_per_capita ($)       generation  suicides_no  suicides/100k pop
0                     796     Generation X           21               6.71
1                     796           Silent           16               5.19
2                     796     Generation X           14               4.83
3                     796  G.I. Generation            1               4.59
4                     796          Boomers            9               3.28
…                   …              …          …                …
27815                2309     Generation X          107               2.96
27816                2309           Silent            9               2.58
27817                2309     Generation Z           60               2.17
27818                2309     Generation Z           44               1.67
27819                2309          Boomers           21               1.46

[27820 rows x 10 columns]
```

```
Summarize dataset:   0%|          | 0/23 [00:00<?, ?it/s]
```

```
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
```

```
Render widgets:   0%|          | 0/1 [00:00<?, ?it/s]
```

```
VBox(children=(Tab(children=(Tab(children=(GridBox(children=(VBox(children=(GridspecLayout(chil
```

```
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

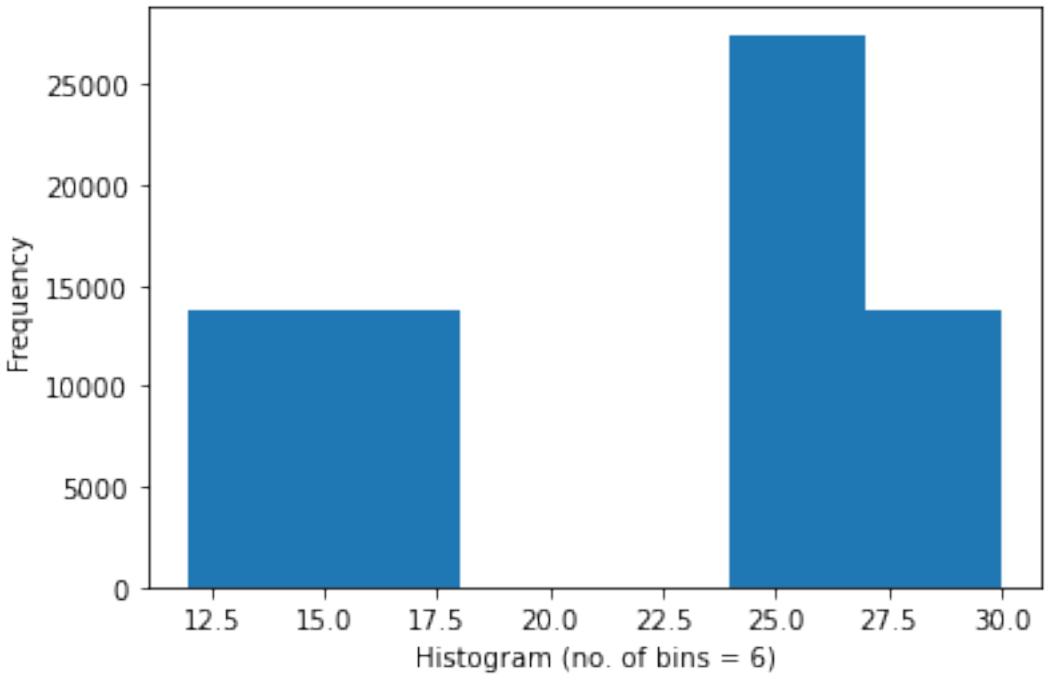
0.1.3 Video Transcoding Dataset

[8]:
```
transcode_meas = pd.read_csv(dataset_folders[2]+"transcoding_mesurment.tsv",␣
↪sep='\t')
transcode_meas = transcode_meas.drop(['id'], axis = 1)
print(transcode_meas)
transcode_meas_prof = ProfileReport(transcode_meas, title="Profile Report for␣
↪Video Transcoding Dataset")
transcode_meas_prof.to_widgets()
transcode_meas_prof.to_file('transcode_meas_prof.html')
```

```
        duration  codec  width  height  bitrate  framerate    i      p  b  \
0      130.35667  mpeg4    176     144    54590  12.000000   27   1537  0
1      130.35667  mpeg4    176     144    54590  12.000000   27   1537  0
2      130.35667  mpeg4    176     144    54590  12.000000   27   1537  0
3      130.35667  mpeg4    176     144    54590  12.000000   27   1537  0
4      130.35667  mpeg4    176     144    54590  12.000000   27   1537  0
…            …      …      …       …        …         …   …      … ..
68779  972.27100   h264    480     360   278822  29.000000  560  28580  0
68780  129.88100    vp8    640     480   639331  30.162790   36   3855  0
68781  249.68000    vp8    320     240   359345  25.068274  129   6113  0
68782  183.62334   h264   1280     720  2847539  29.000000   98   5405  0
68783  294.61334  mpeg4    176     144    55242  12.000000   61   3474  0

       frames   i_size    p_size  b_size      size o_codec  o_bitrate  \
0        1564    64483    825054       0    889537   mpeg4      56000
1        1564    64483    825054       0    889537   mpeg4      56000
2        1564    64483    825054       0    889537   mpeg4      56000
3        1564    64483    825054       0    889537   mpeg4      56000
4        1564    64483    825054       0    889537   mpeg4      56000
…         …       …         …       …         …       …          …
68779   29140  7324628  26561730       0  33886358     flv     242000
68780    3891   875784   9503846       0  10379630   mpeg4     539000
68781    6242  1758664   9456514       0  11215178     flv     539000
68782    5503  5246294  60113035       0  65359329   mpeg4     539000
68783    3535    84002   1950409       0   2034411    h264     820000

       o_framerate  o_width  o_height    umem   utime
0            12.00      176       144   22508   0.612
1            12.00      320       240   25164   0.980
2            12.00      480       360   29228   1.216
3            12.00      640       480   34316   1.692
4            12.00     1280       720   58528   3.456
…               …        …         …       …
68779        24.00      640       480   88692   1.552
68780        29.97     1920      1080  107524  18.557
68781        12.00      176       144   88708   0.752
68782        12.00      320       240   88724   5.444
```

```
68783        24.00       176        144    88736    3.076

[68784 rows x 21 columns]

Summarize dataset:   0%|          | 0/34 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render widgets:   0%|          | 0/1 [00:00<?, ?it/s]

VBox(children=(Tab(children=(Tab(children=(GridBox(children=(VBox(children=(GridspecLayout(chil

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]

Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

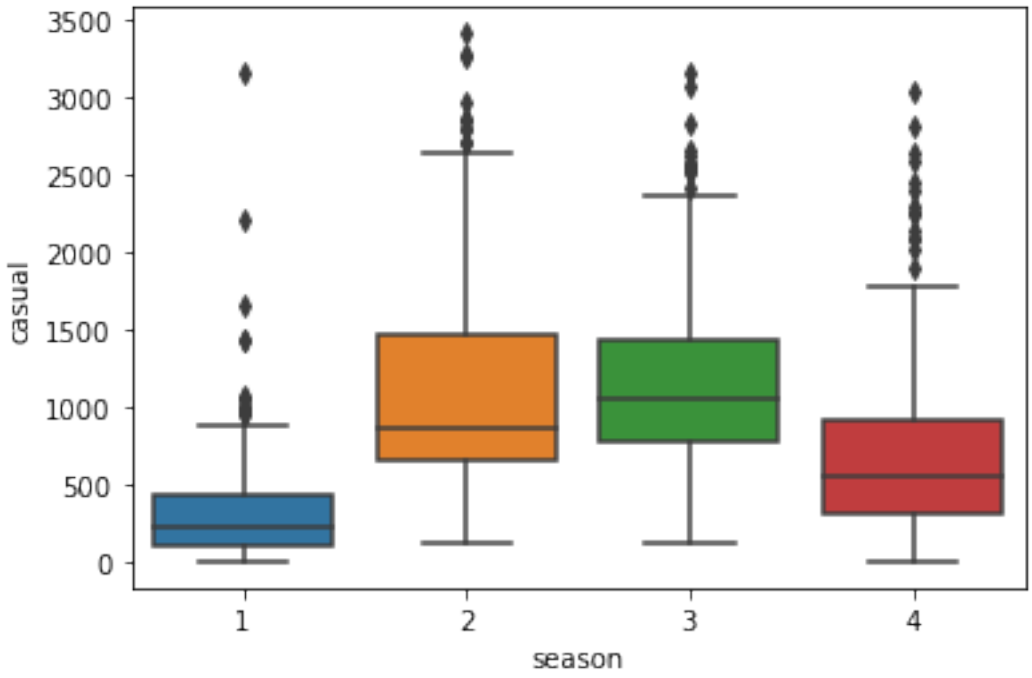
[10]:
```
import matplotlib.pyplot as plt
x = [15.0, 12.0, 29.97,25.0,24.0]
y = [13772, 13764, 13759, 13751, 13738]
plt.hist(x,6, weights=y)
plt.xlabel("Histogram (no. of bins = 6)")
plt.ylabel("Frequency")
plt.savefig("Q2c_o_framerate",dpi=300,bbox_inches='tight')
plt.show()
```

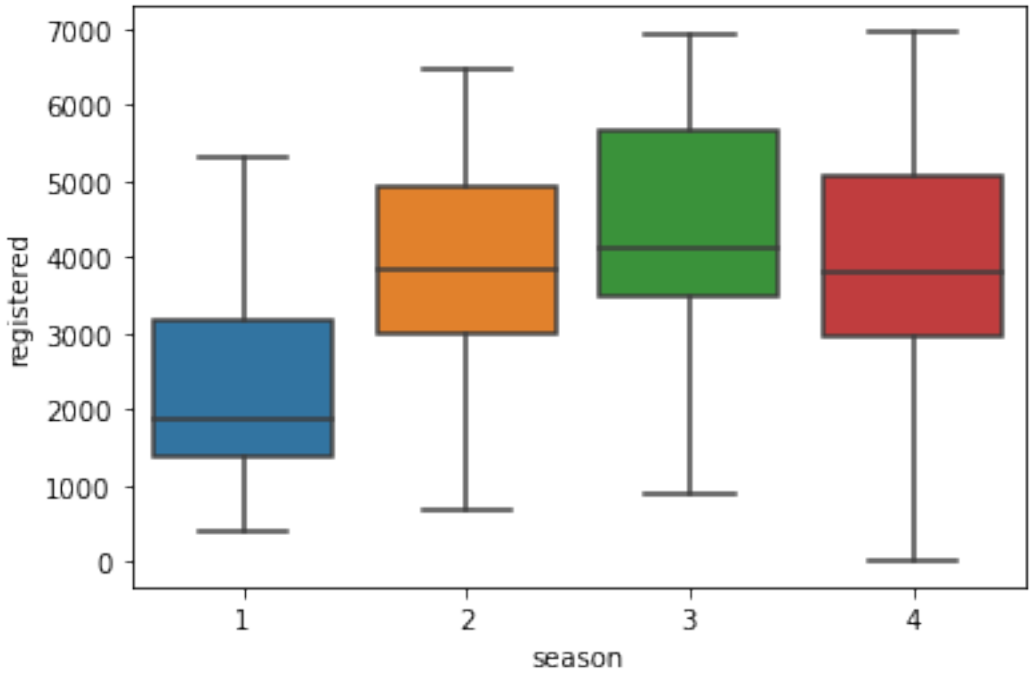


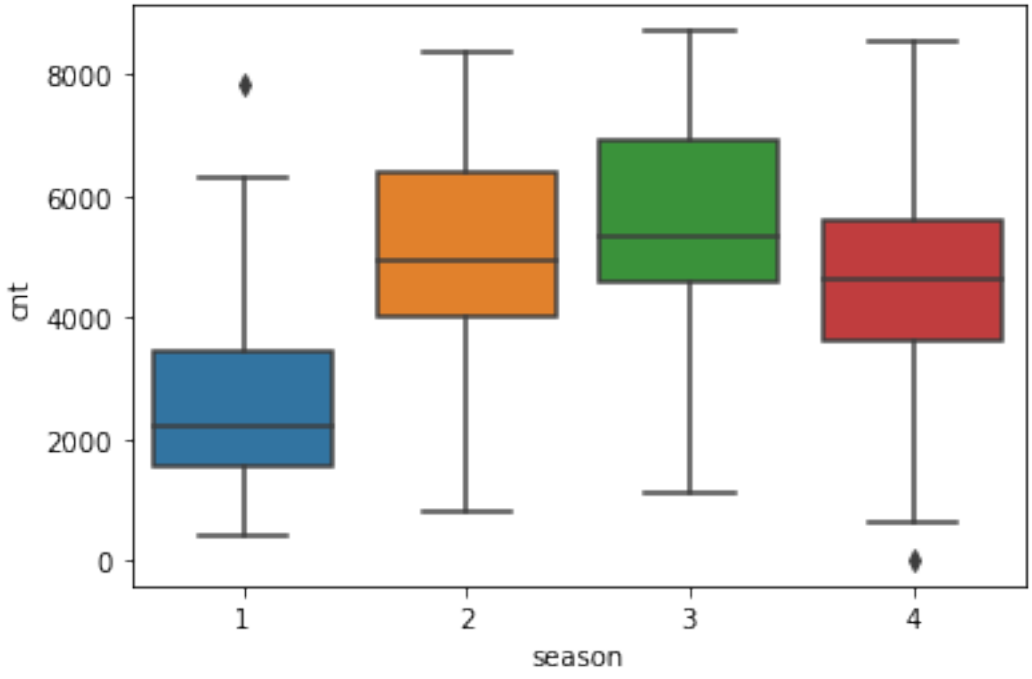
0.2 Question 3

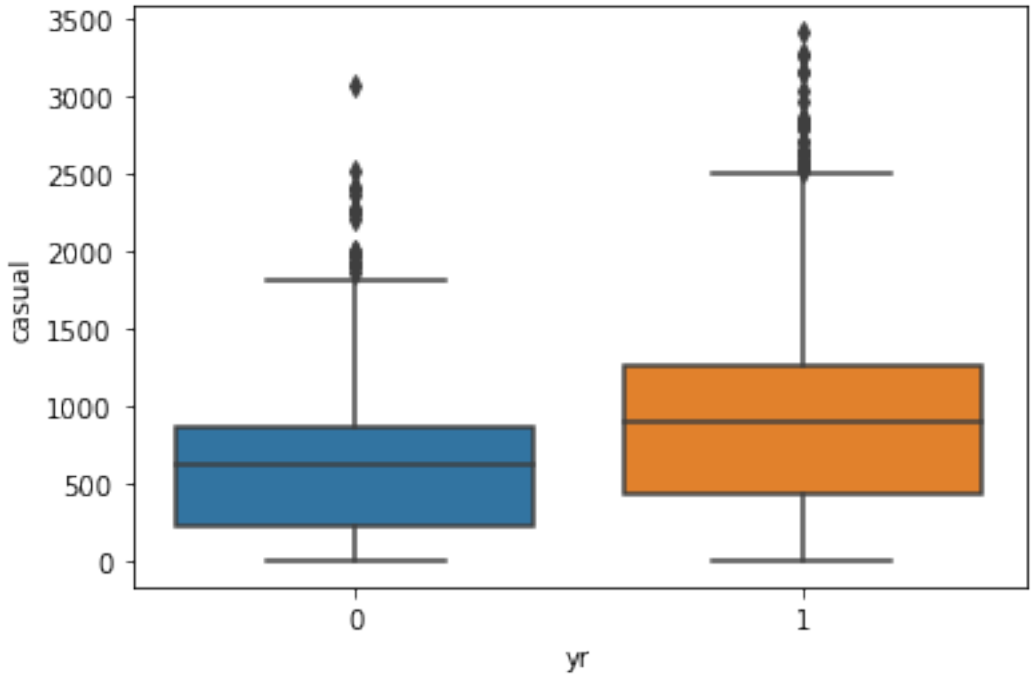
0.2.1 Bike Sharing Dataset

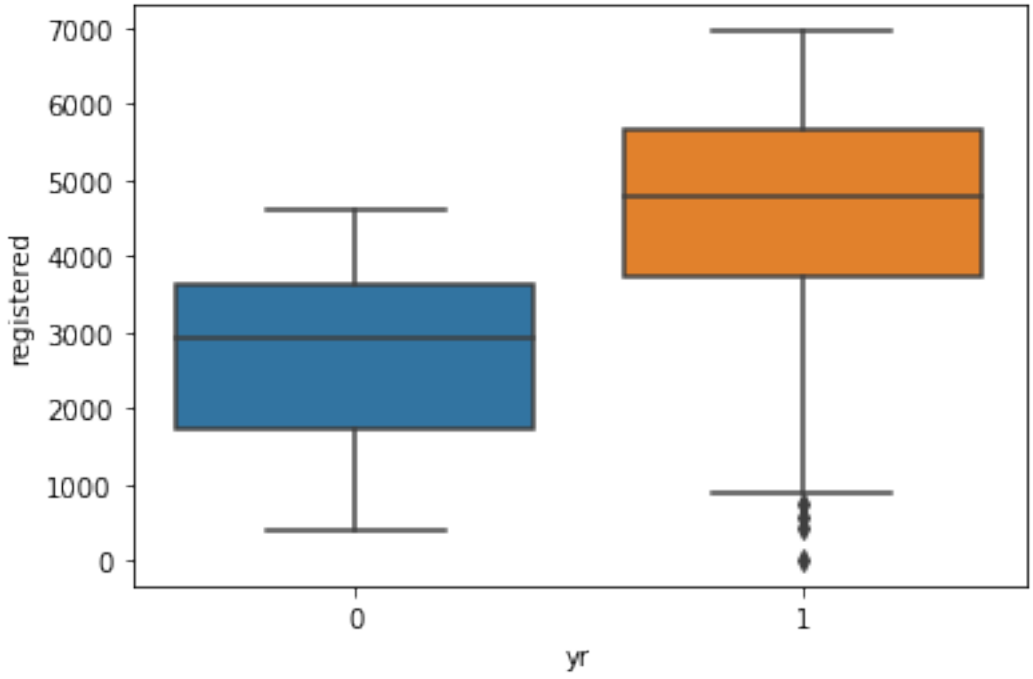
[625]:
```
cat_list = ['season','yr','mnth','holiday','weekday','workingday', 'weathersit']
target_var = ['casual','registered','cnt']
for item in cat_list:
    for target in target_var:
        seaborn.boxplot(x = bike[item],y = bike[target],␣
↪order=list(set(bike[item])))
        plt.savefig('Q3a'+target+'_'+item+'.png',dpi=300,bbox_inches='tight')
        plt.show()
```

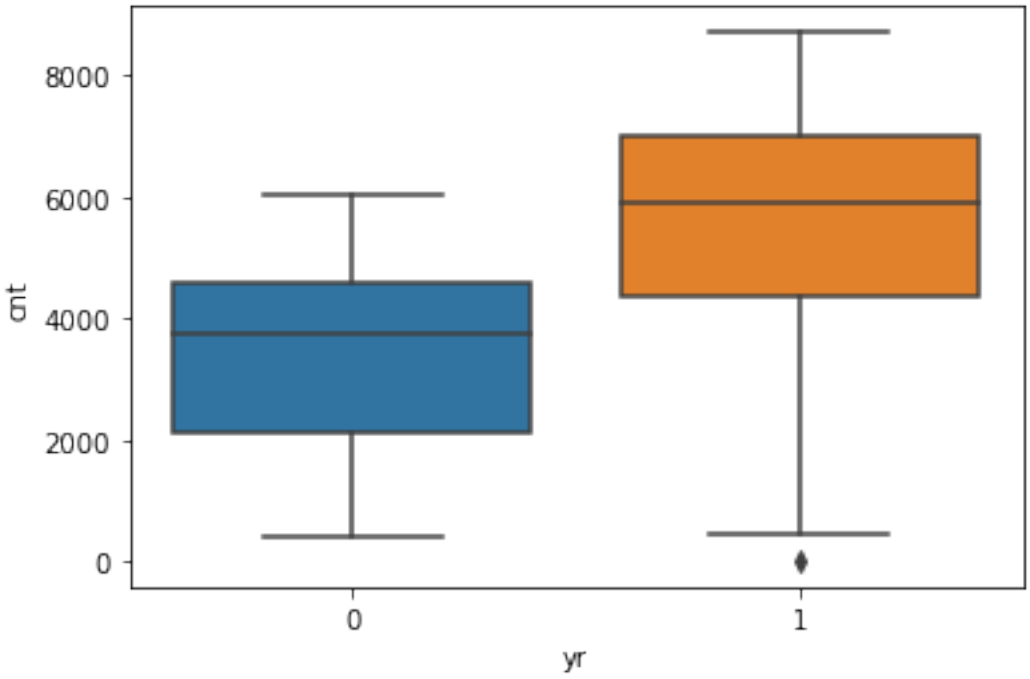


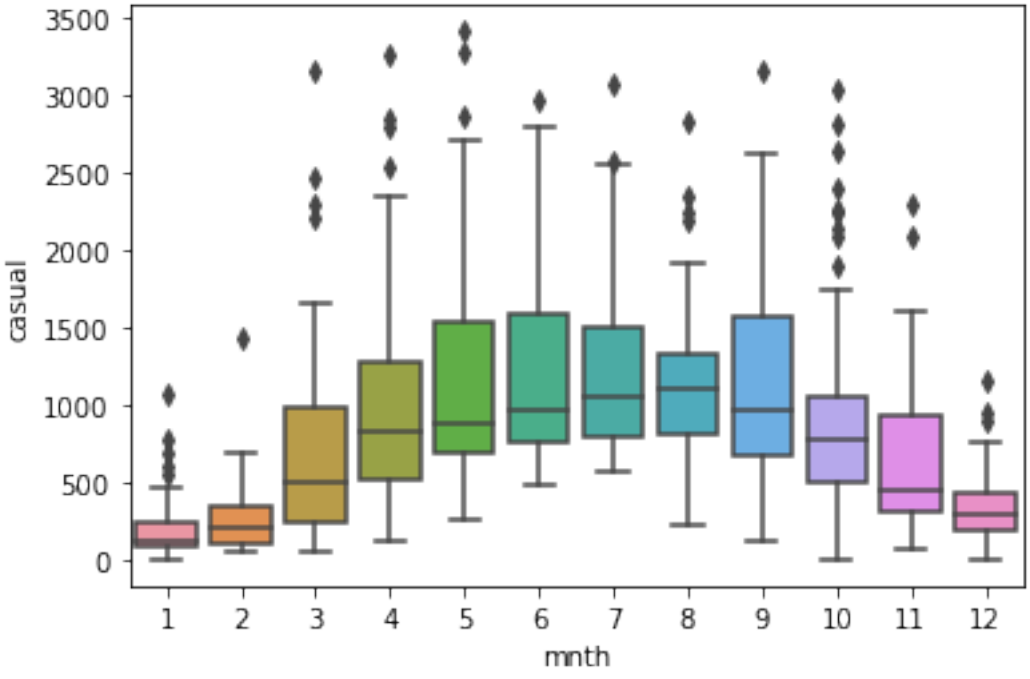

registered
7000
6000
5000
4000
3000
2000
1000
0
1
2
3
4
season

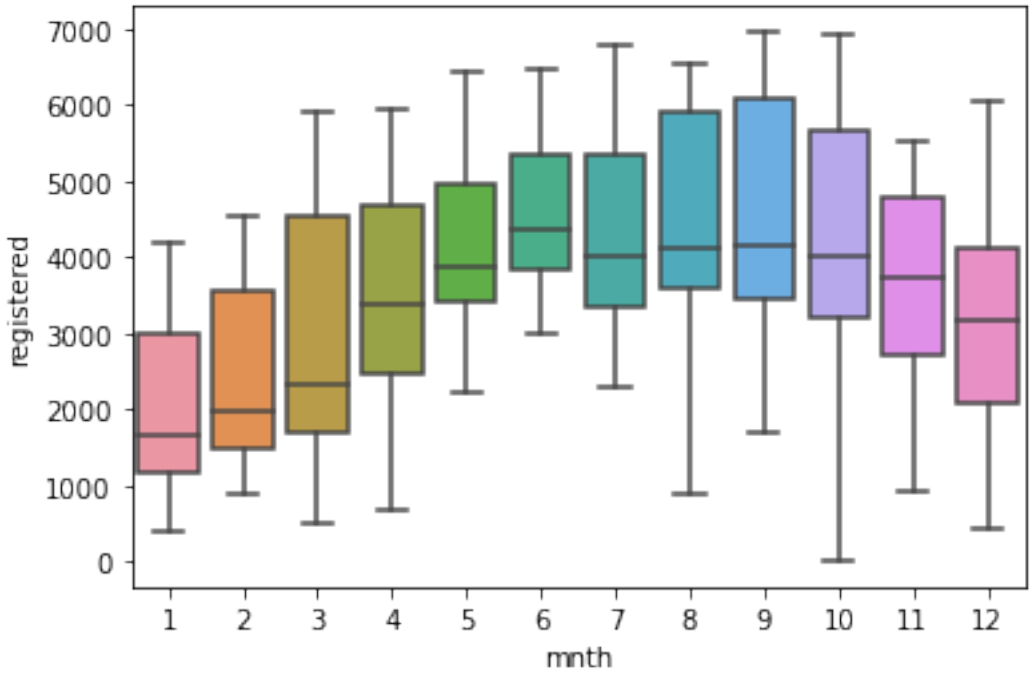

cnt
8000
6000
4000
2000
0
1
2
3
4
season

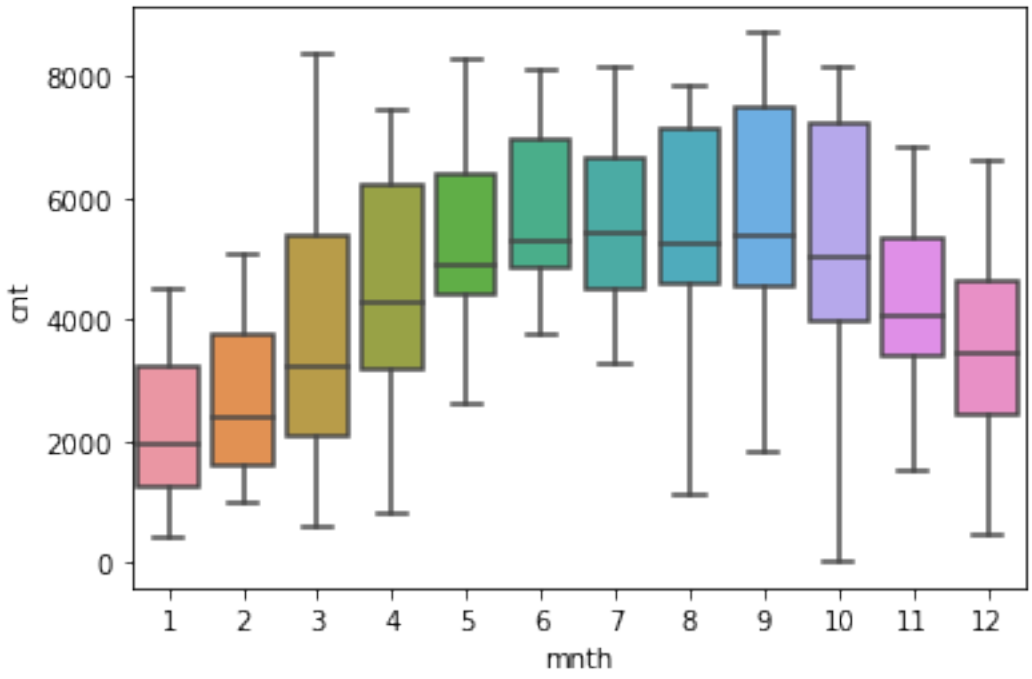

3500
3000
2500
2000
1500
1000
500
0
casual
0
1
yr

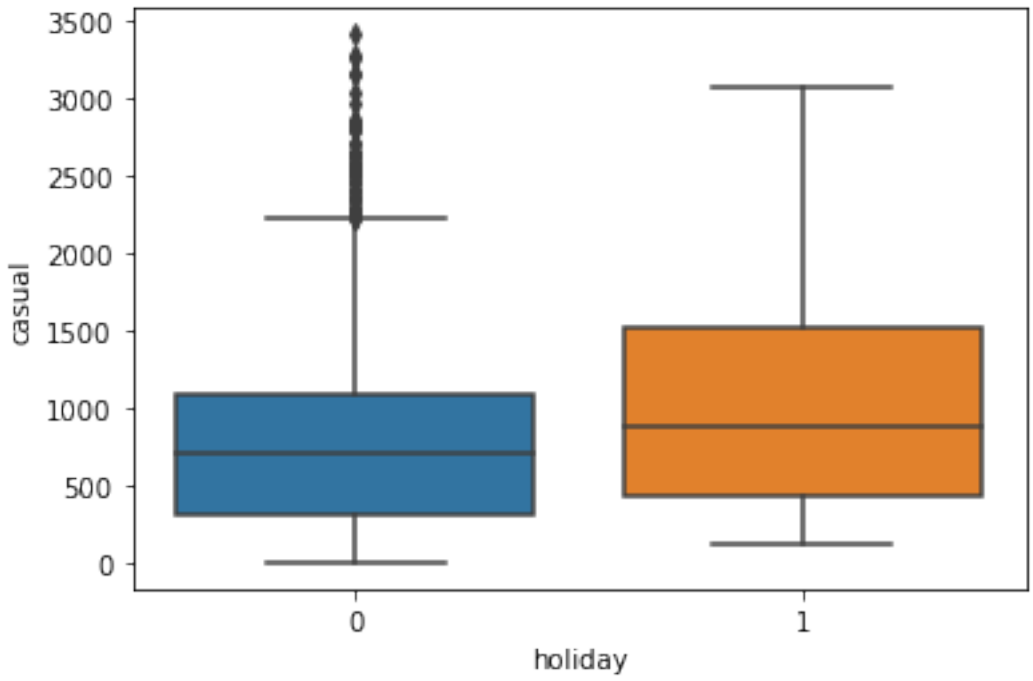

7000
6000
5000
4000
3000
2000
1000
0
registered
0
1
yr

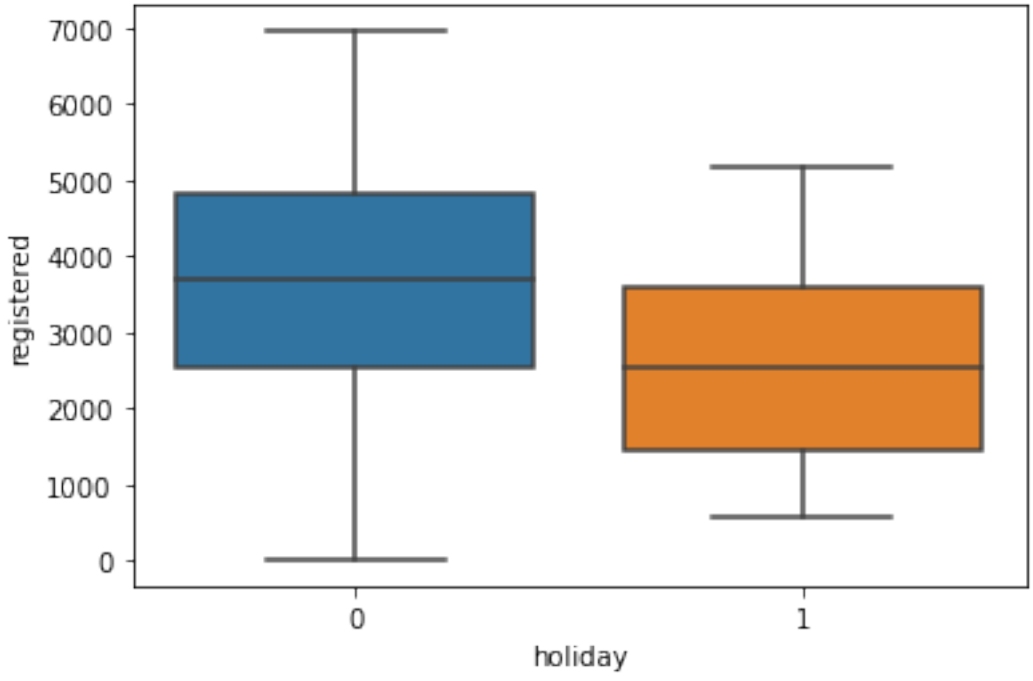

8000
6000
4000
2000
0
cnt
0
1
yr

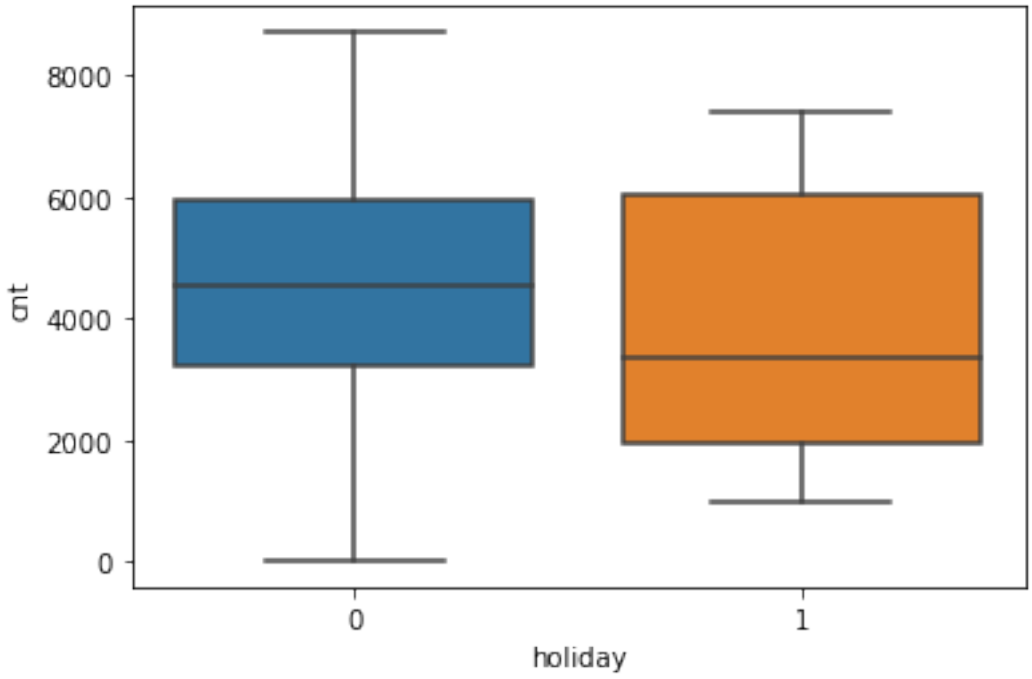

3500
3000
2500
2000
1500
1000
500
0
casual
1
2
3
4
5
6
7
8
9
10
11
12
mnth

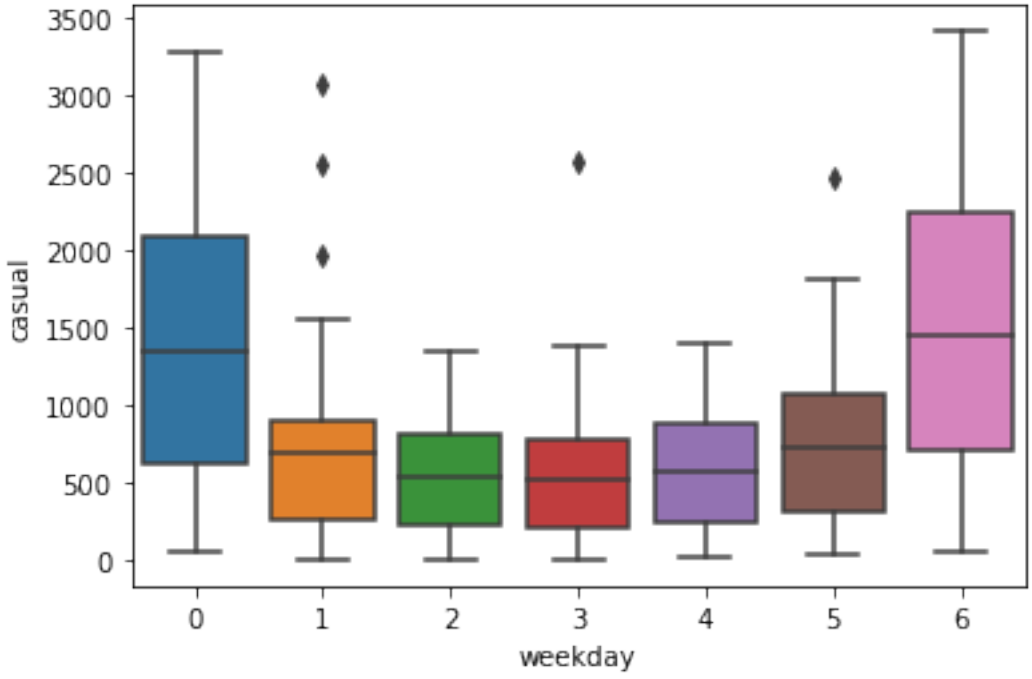

registered
7000
6000
5000
4000
3000
2000
1000
0
1
2
3
4
5
6
7
8
9
10
11
12
mnth

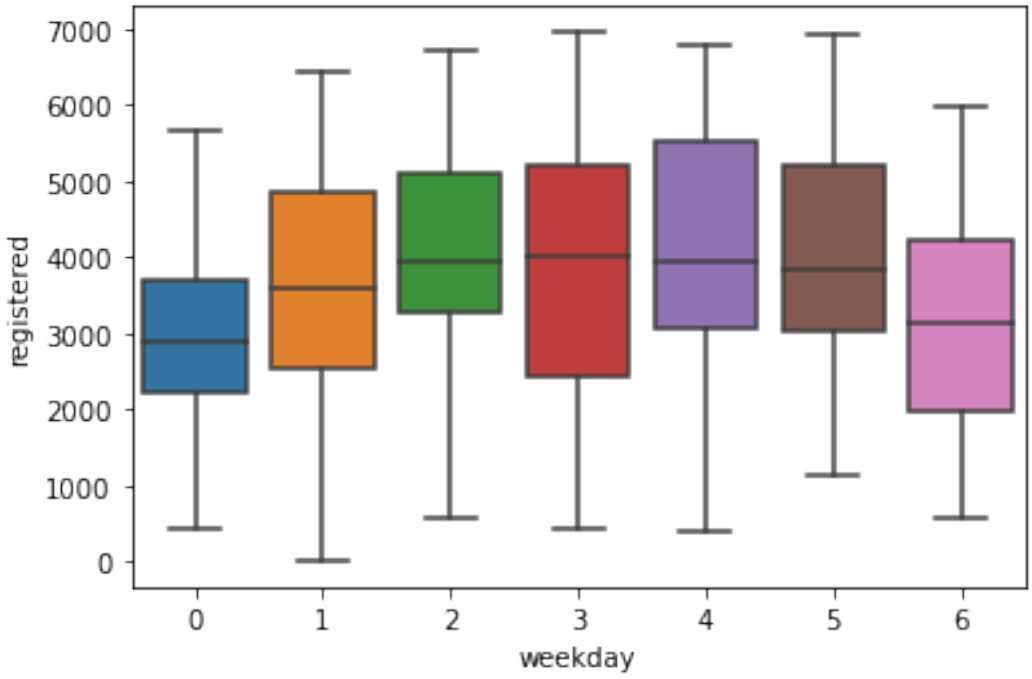

cnt
8000
6000
4000
2000
0
1
2
3
4
5
6
7
8
9
10
11
12
mnth

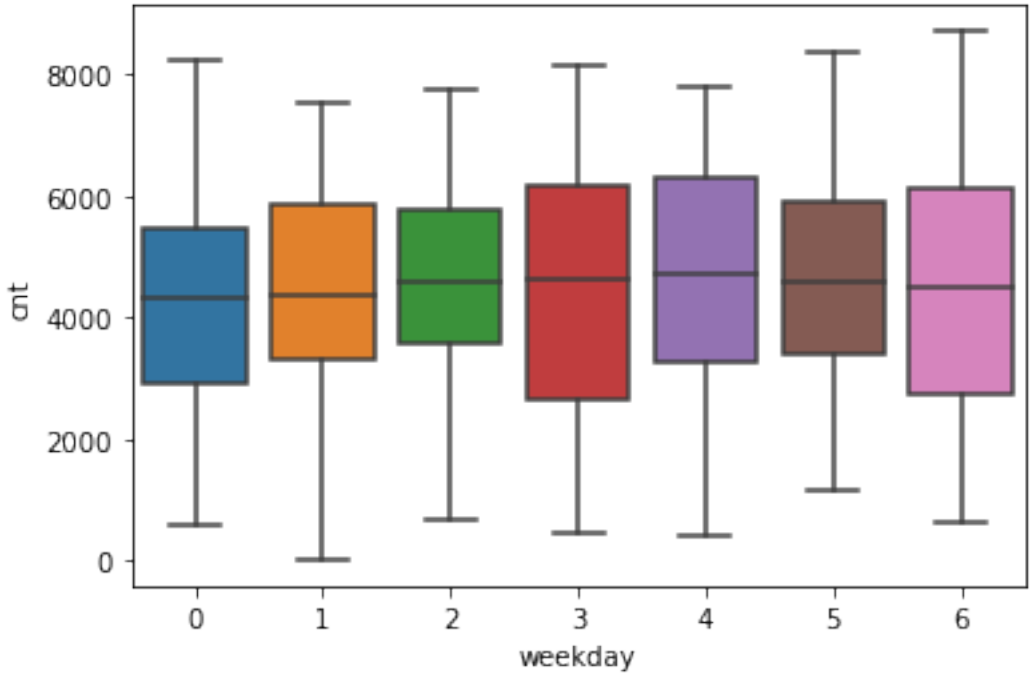

3500
3000
2500
2000
1500
1000
500
0
casual
0
1
holiday

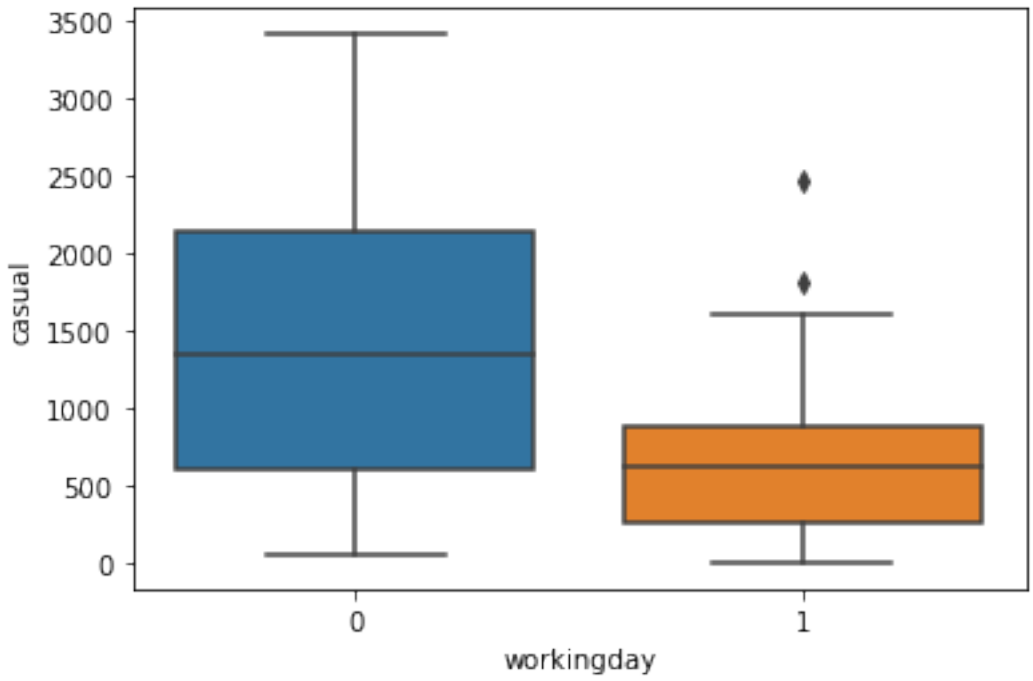

7000
6000
5000
4000
3000
2000
1000
0
registered
0
1
holiday

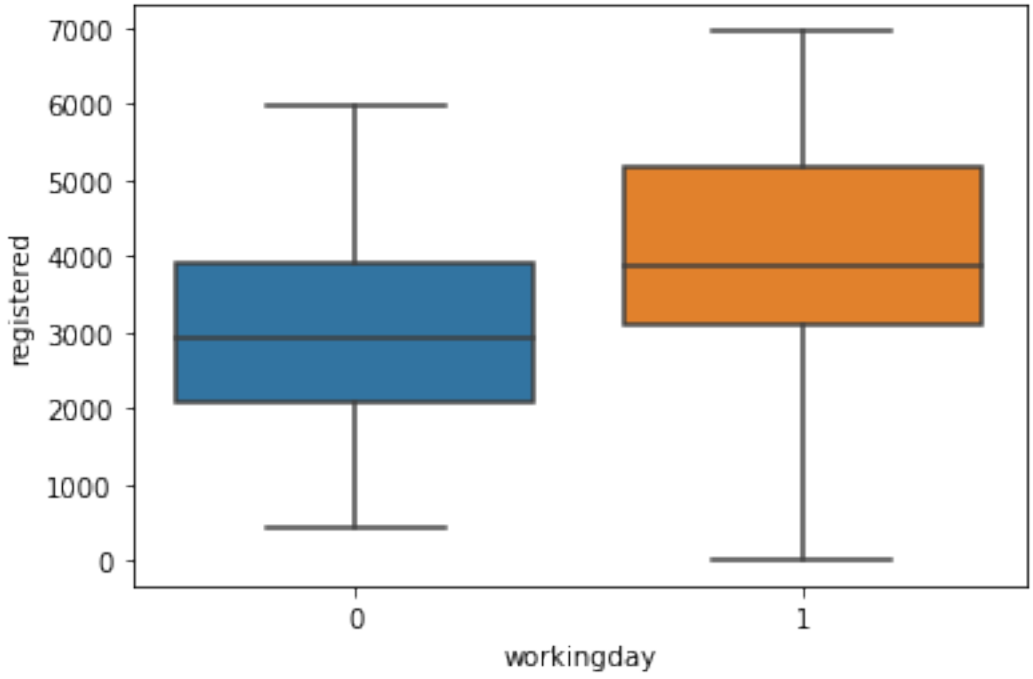

8000
6000
4000
2000
0
cnt
0
1
holiday

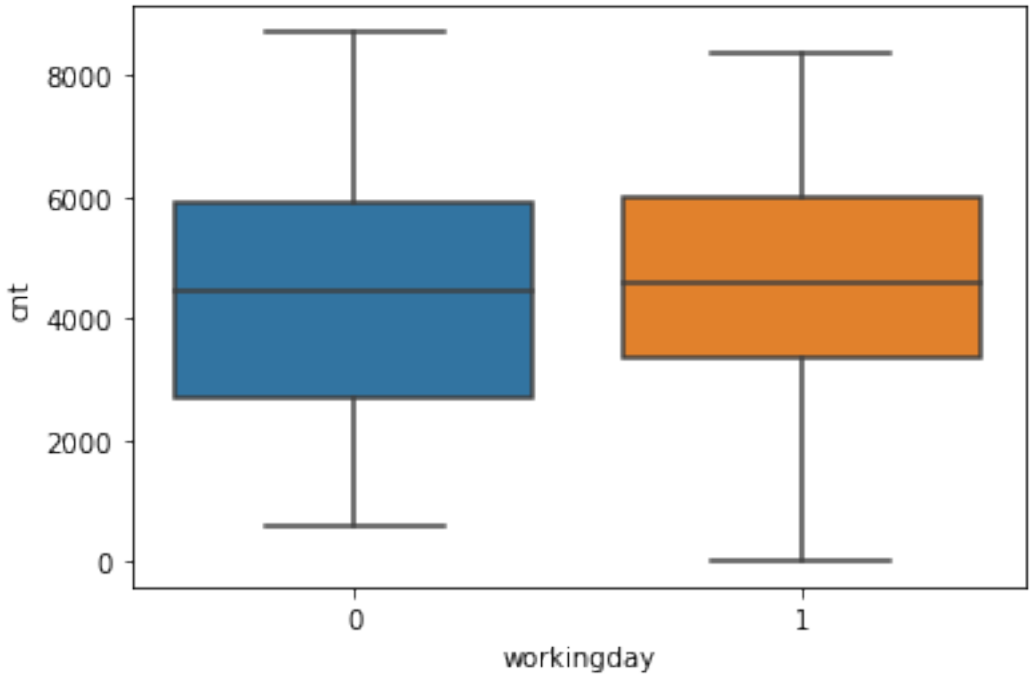

3500
3000
2500
2000
1500
1000
500
0
casual
0
1
2
3
4
5
6
weekday

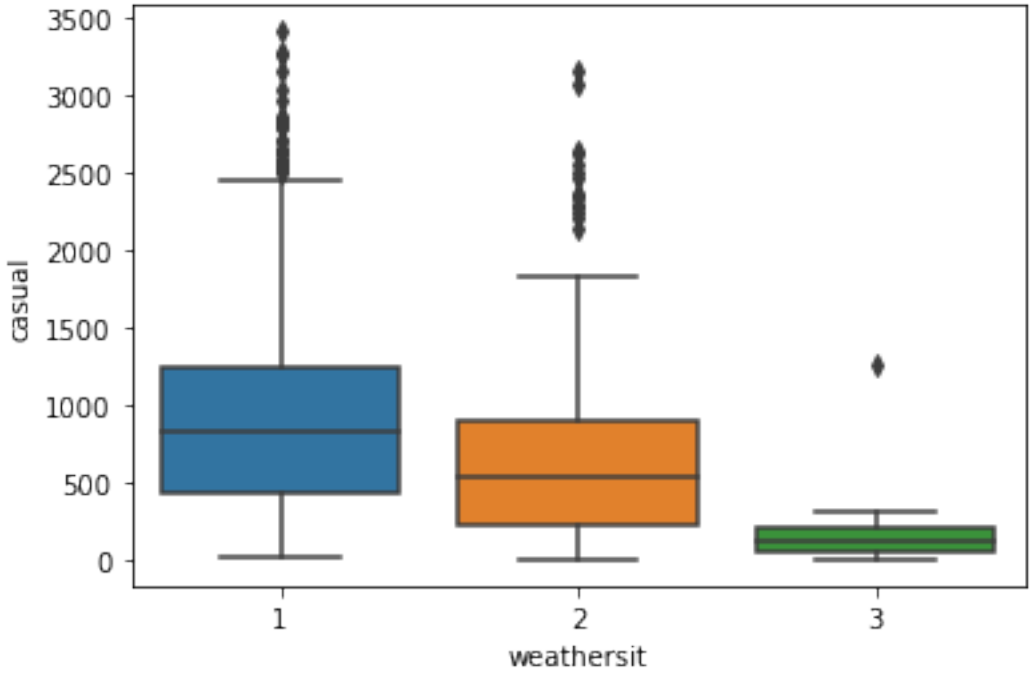

registered
7000
6000
5000
4000
3000
2000
1000
0
0
1
2
3
4
5
6
weekday

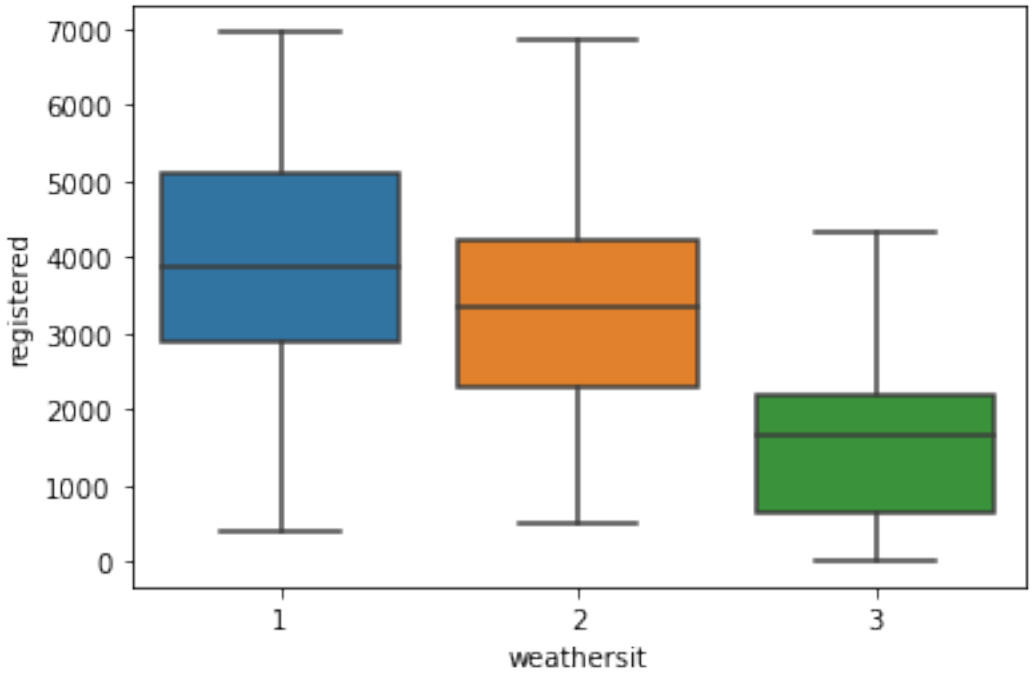

cnt
8000
6000
4000
2000
0
0
1
2
3
4
5
6
weekday

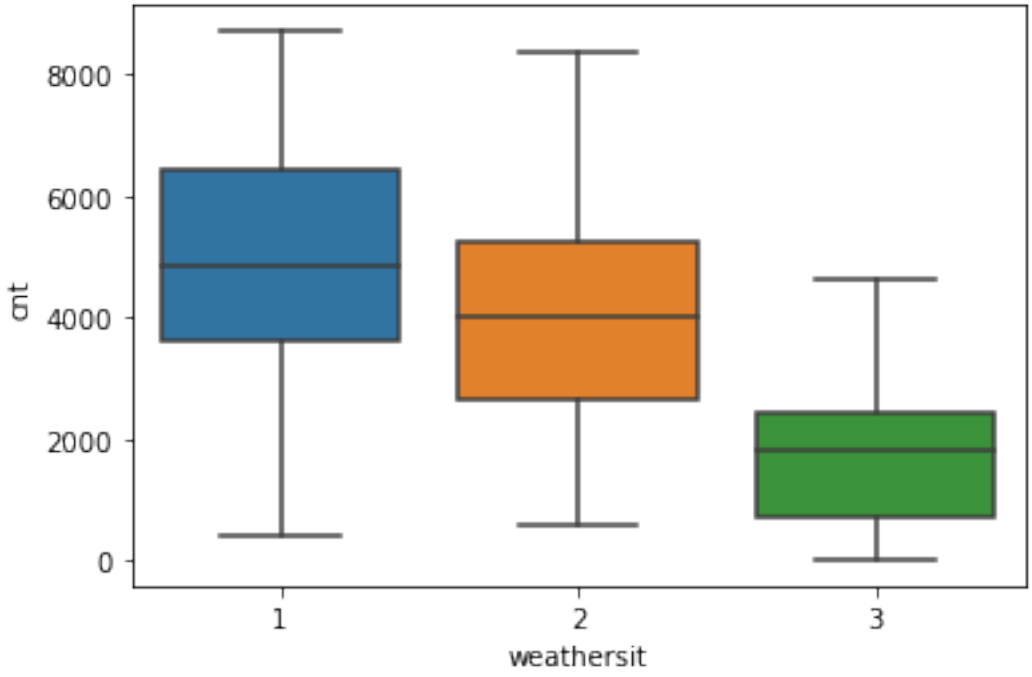

3500
3000
2500
2000
1500
1000
500
0
casual
0
1
workingday


7000
6000
5000
4000
3000
2000
1000
0
registered
0
1
workingday


8000
6000
4000
2000
0
cnt
0
1
workingday

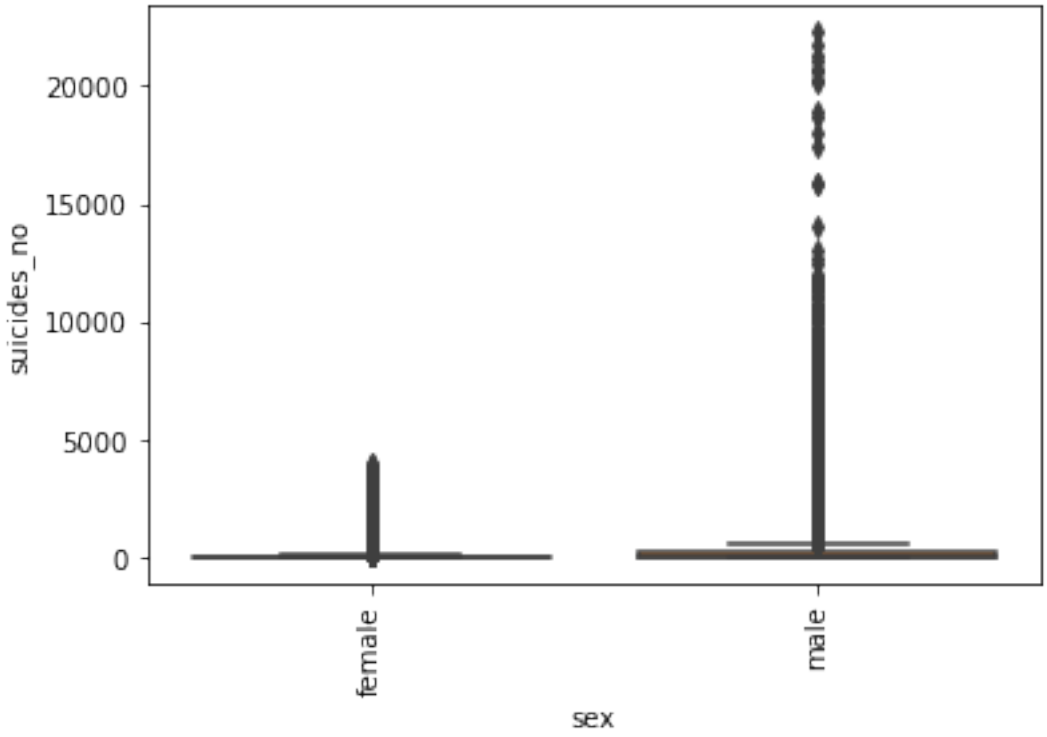

3500
3000
2500
2000
1500
1000
500
0
casual
1
2
3
weathersit

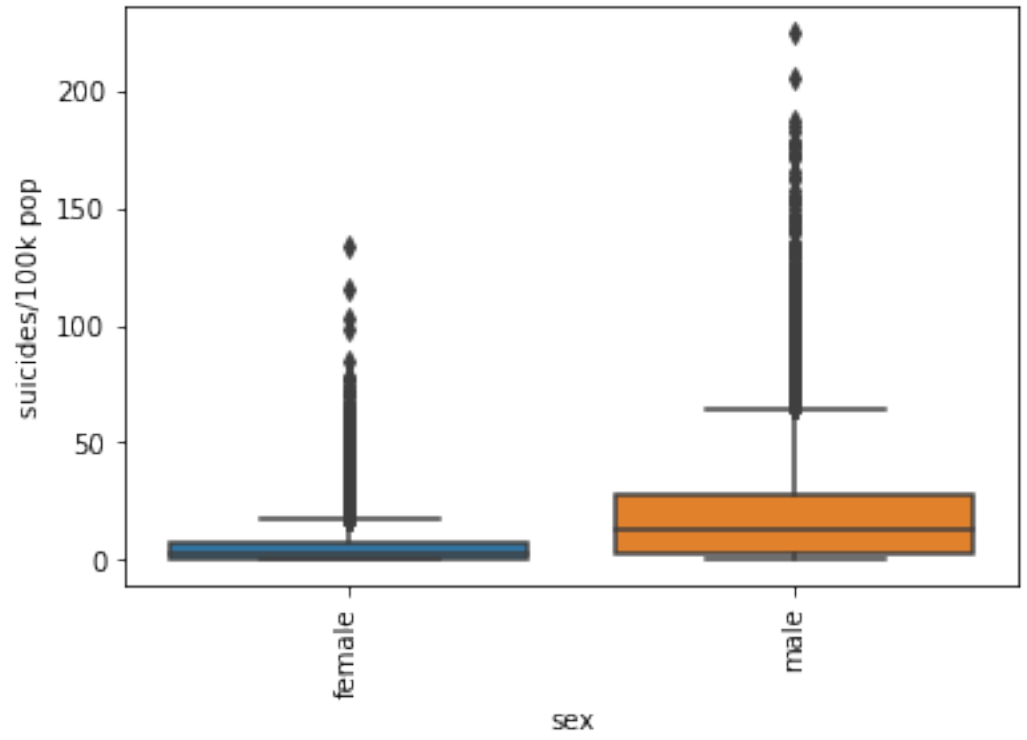

registered
7000
6000
5000
4000
3000
2000
1000
0
1
2
3
weathersit

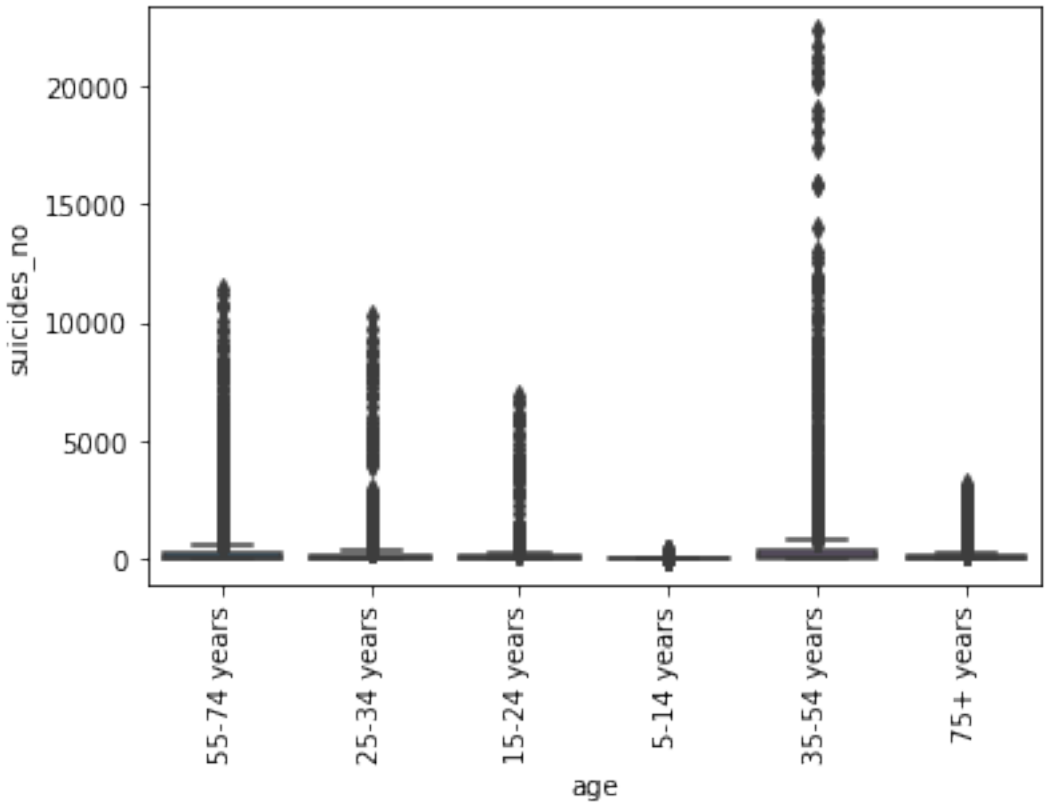

cnt
8000
6000
4000
2000
0
1
2
3
weathersit

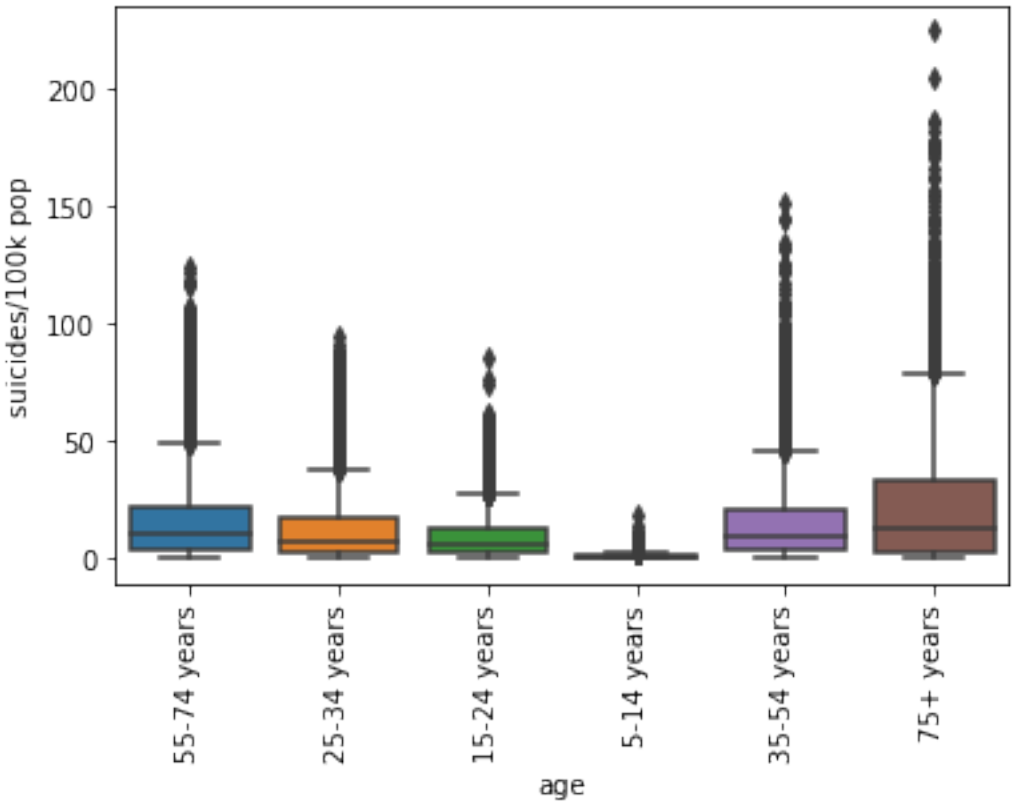
0.2.2 Suicide Rates Dataset

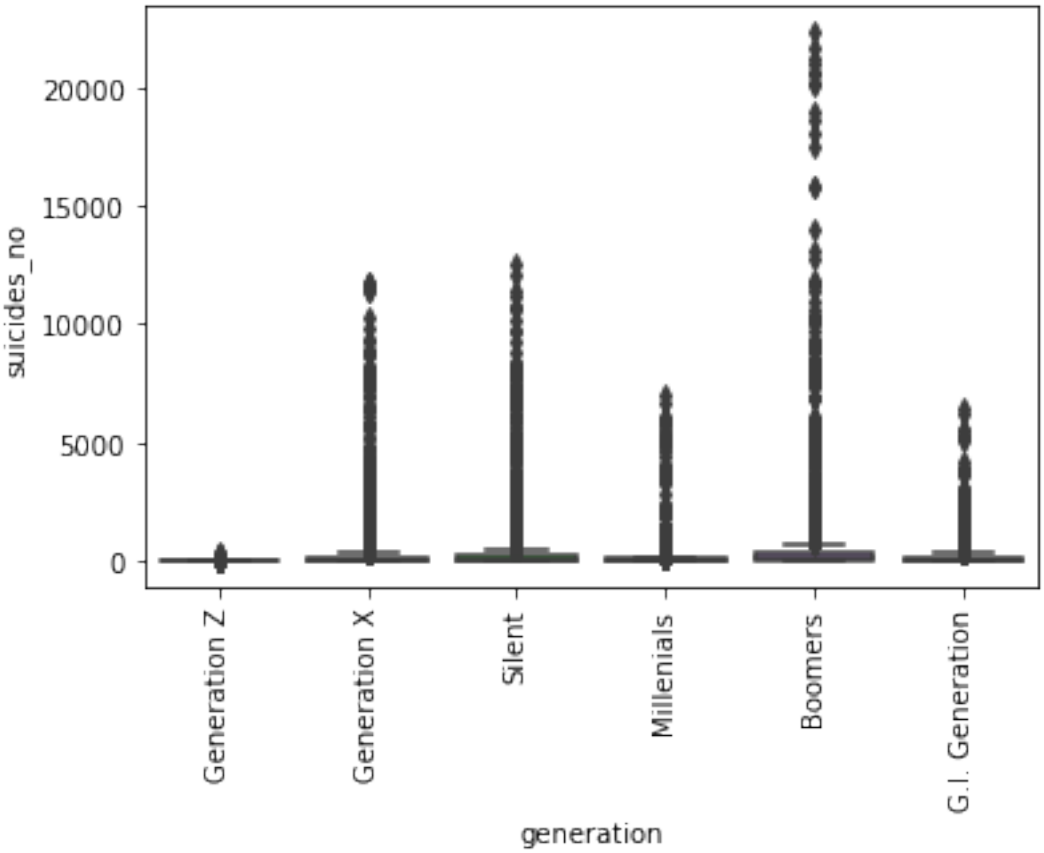
[626]:
```
cat_list = ['sex','age','generation']
target_var = ['suicides_no','suicides/100k pop']
for item in cat_list:
    for target in target_var:
        ax = seaborn.boxplot(x = suicide[item],y = suicide[target],␣
↪order=list(set(suicide[item])))
        plt.setp(ax.get_xticklabels(), rotation=90)
        plt.savefig('Q3b'+target.replace('/','_')+'_'+item+'.
↪png',dpi=300,bbox_inches='tight')
        plt.show()
```

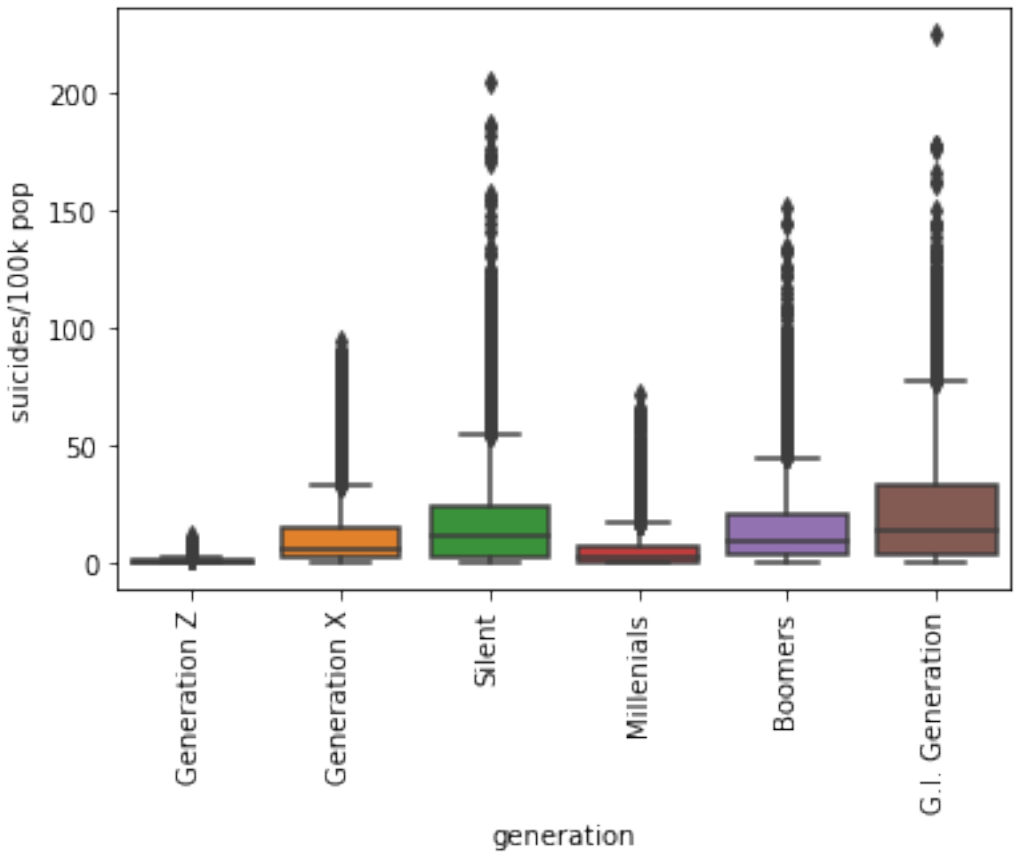



suicides/100k pop
200
150
100
50
0
female
male
sex


suicides_no
20000
15000
10000
5000
0
55-74 years
25-34 years
15-24 years
5-14 years
35-54 years
75+ years
age

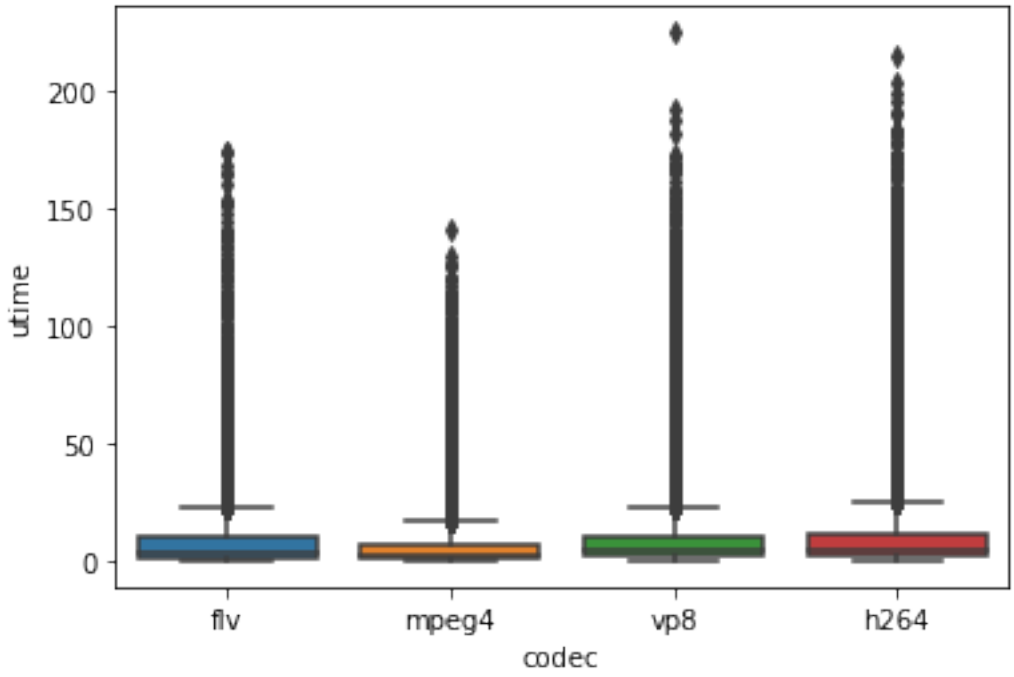

suicides/100k pop
200
150
100
50
0
55-74 years
25-34 years
15-24 years
5-14 years
35-54 years
75+ years
age

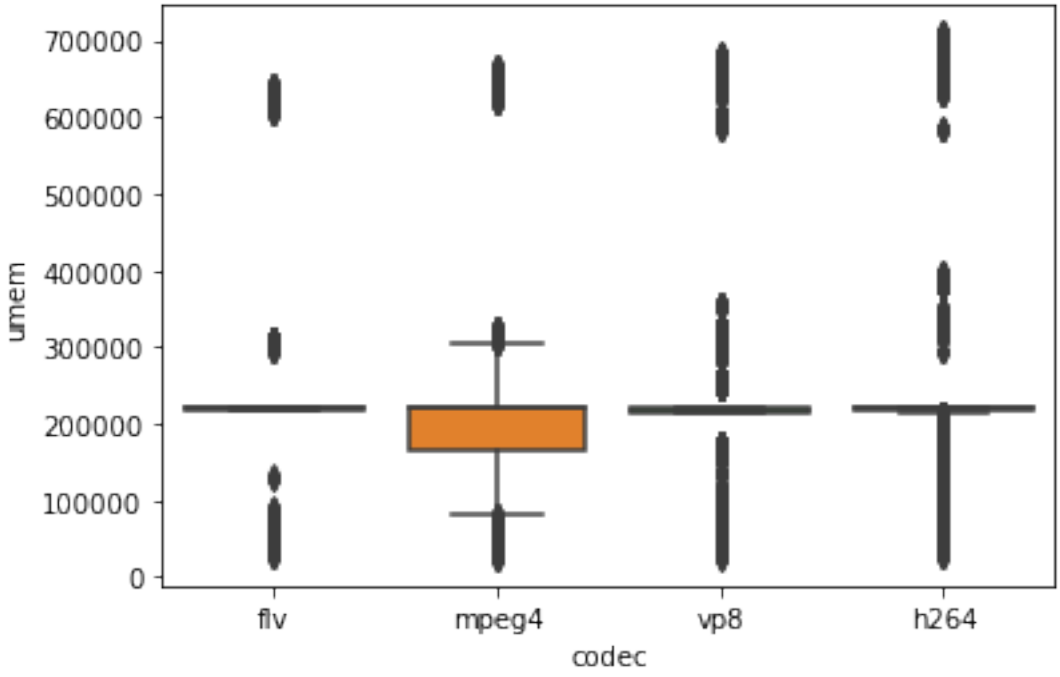

20000
15000
10000
5000
0
suicides_no
Generation Z
Generation X
Silent
Millenials
Boomers
G.I. Generation
generation

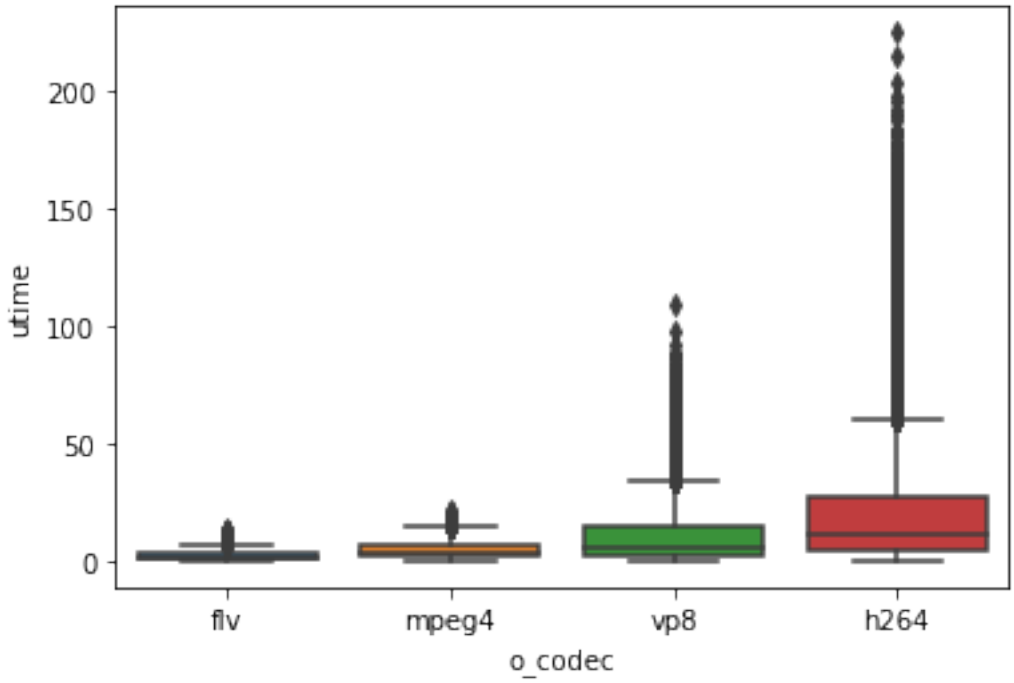


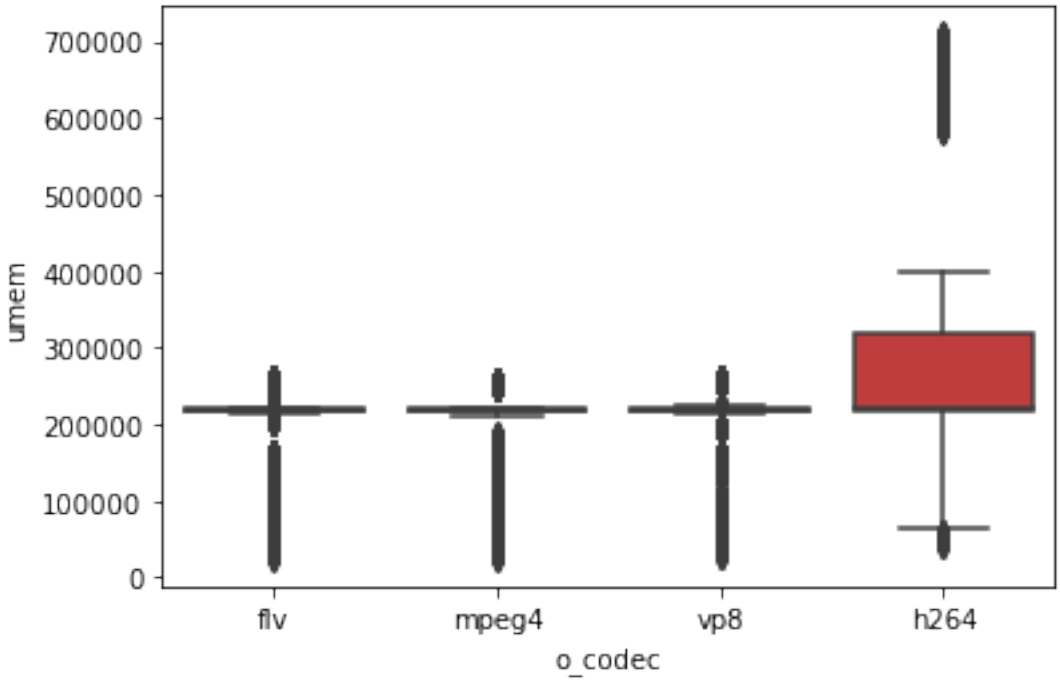
0.2.3 Video Transcoding Dataset

[9]:
```
cat_list = ['codec','o_codec']
target_var = ['utime','umem']
for item in cat_list:
    for target in target_var:
        seaborn.boxplot(x = transcode_meas[item],y = transcode_meas[target],␣
↪order=list(set(transcode_meas[item])))
        plt.savefig('Q3c'+target+'_'+item+'.png',dpi=300,bbox_inches='tight')
        plt.show()
```


utime
200
150
100
50
0
flv
mpeg4
vp8
h264
codec

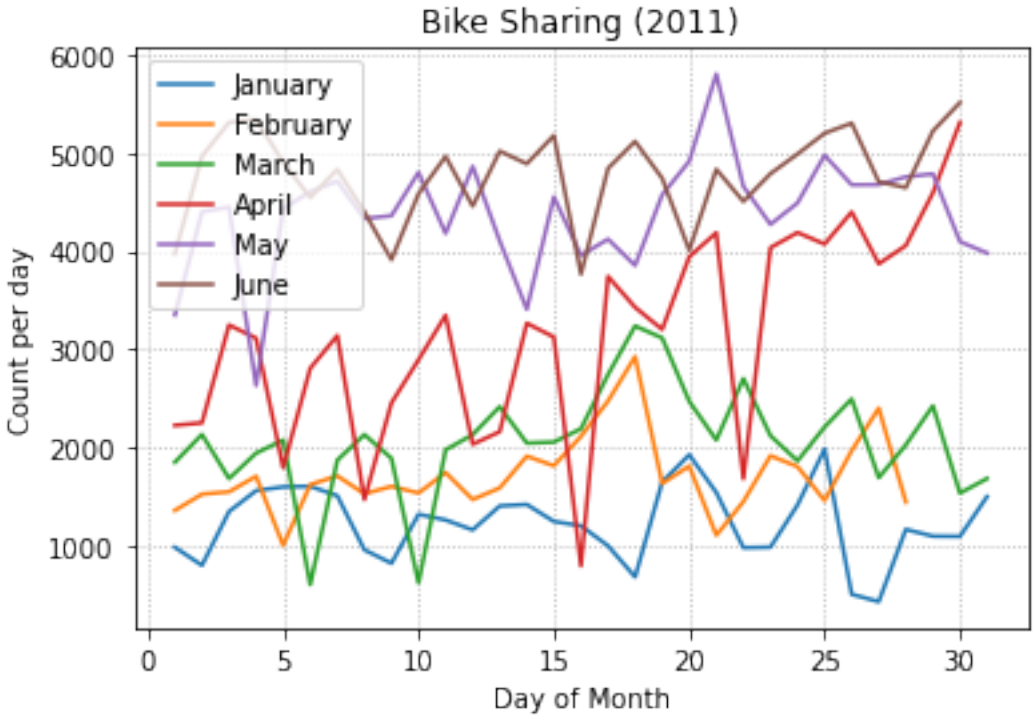

umem
700000
600000
500000
400000
300000
200000
100000
0
flv
mpeg4
vp8
h264
codec


utime
200
150
100
50
0
flv
mpeg4
vp8
h264
o_codec


umem
700000
600000
500000
400000
300000
200000
100000
0
flv
mpeg4
vp8
h264
o_codec

0.3 Question 4

[628]:
```
plt.plot(np.arange(1,32,1),bike['cnt'][0:31])
plt.plot(np.arange(1,29,1),bike['cnt'][31:31+28])
plt.plot(np.arange(1,32,1),bike['cnt'][31+28:31+28+31])
plt.plot(np.arange(1,31,1),bike['cnt'][31+28+31:31+28+31+30])
plt.plot(np.arange(1,32,1),bike['cnt'][31+28+31+30:31+28+31+30+31])
plt.plot(np.arange(1,31,1),bike['cnt'][31+28+31+30+31:31+28+31+30+31+30])
plt.legend(['January','February','March','April','May','June'],loc='best')
plt.grid(linestyle=':')
plt.xlabel('Day of Month')
plt.ylabel('Count per day')
plt.title('Bike Sharing (2011)')
plt.savefig('Q4.png',dpi=300,bbox_inches='tight')
plt.show()
```

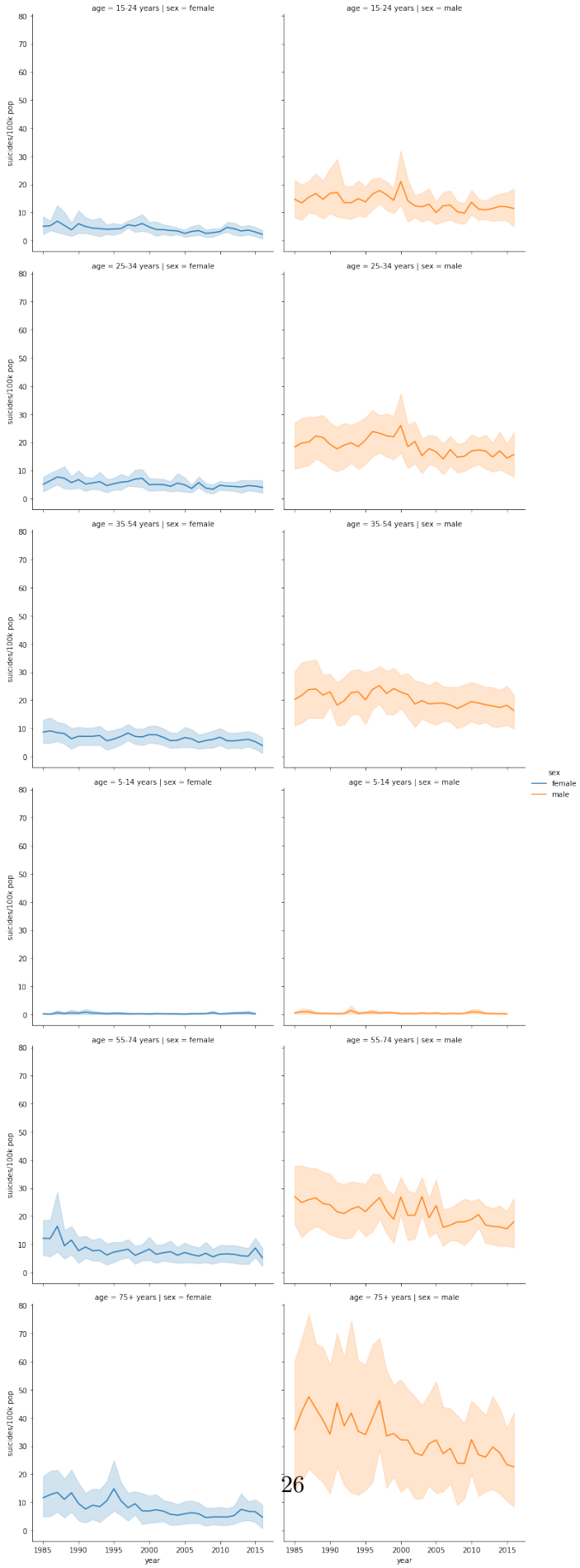


0.4 Question 5

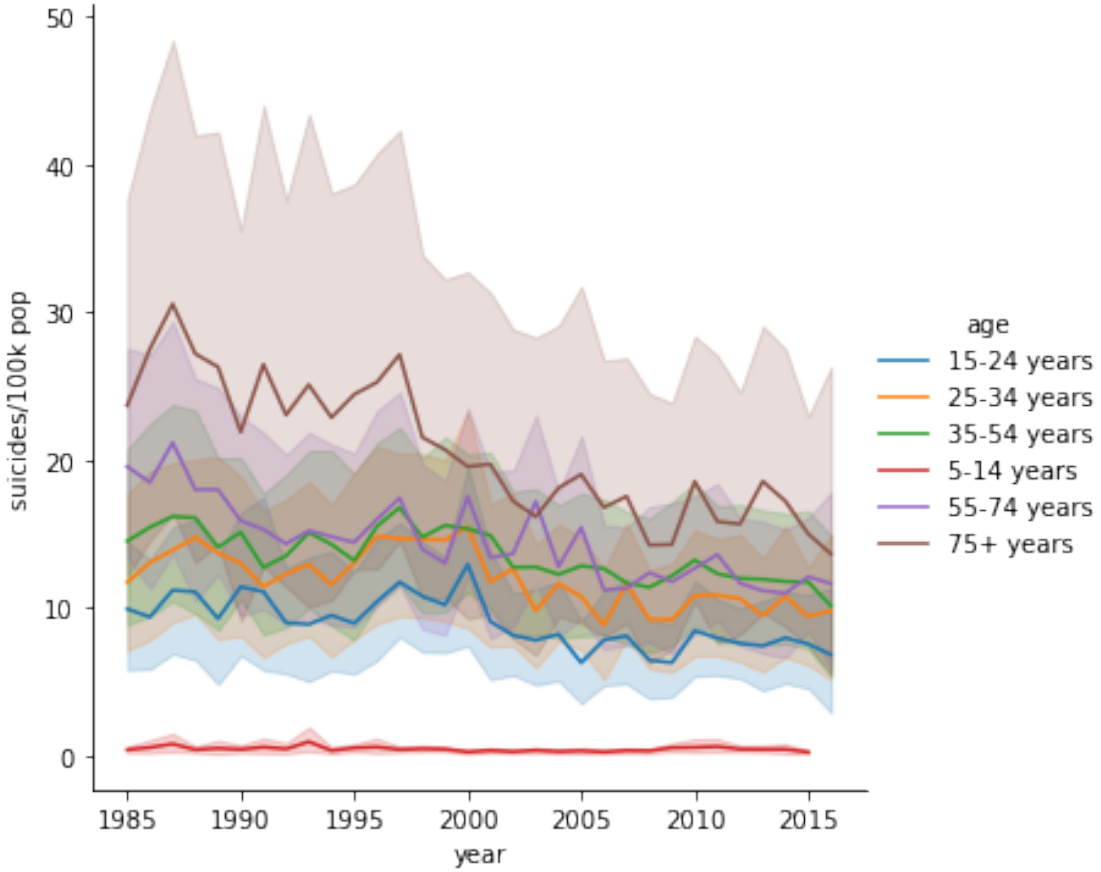
[629]:
```
country_list = suicide.country.unique()
time_span = []
for item in country_list:
    idx = suicide.country[suicide.country == item].index.tolist()
    idx_range = idx[::len(idx)-1]
```

```
    time_span.append(suicide.year[idx_range[1]]-suicide.year[idx_range[0]])
target_country = country_list[sorted(range(len(time_span)), key=lambda i:␣
 ↪time_span[i], reverse=True)[:10]]
new_df = suicide[suicide['country'].isin(target_country)]
```

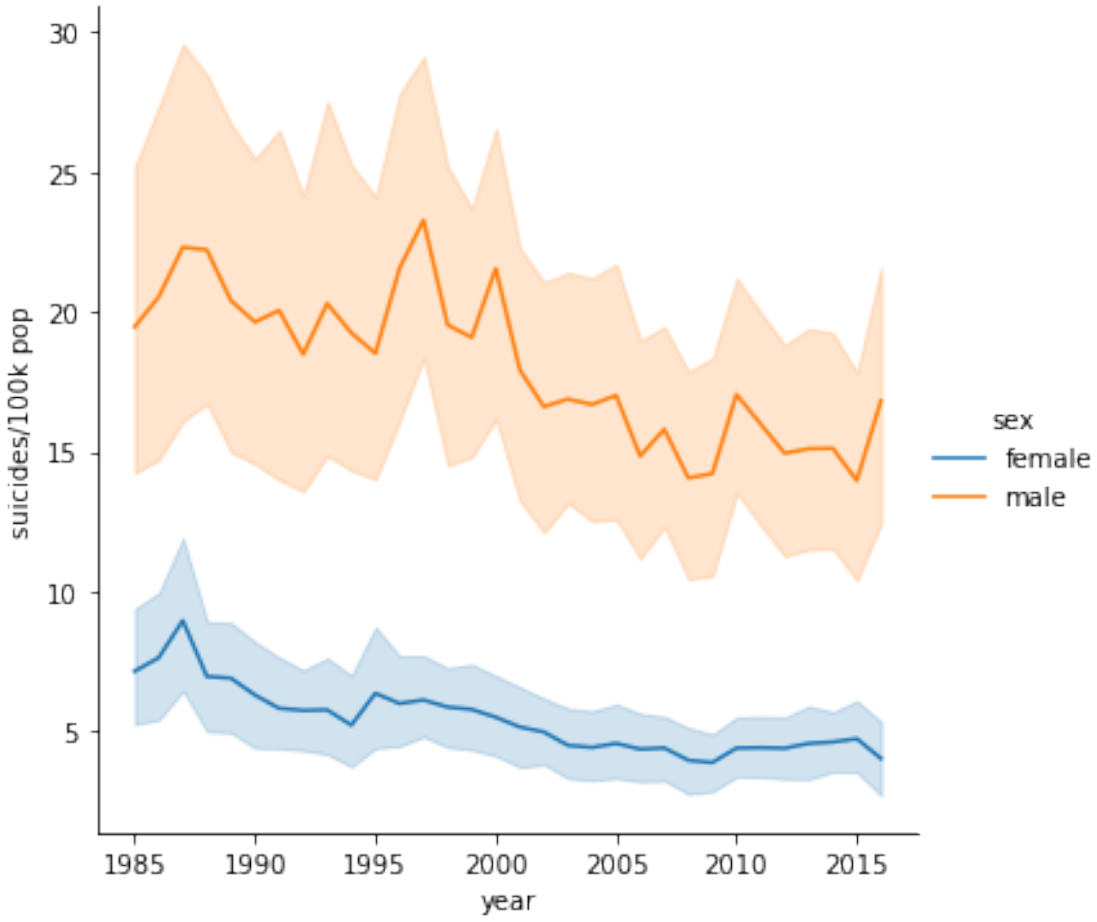
[634]:
```
seaborn.relplot(data=new_df, x="year", y="suicides/100k␣
 ↪pop",hue="sex",kind="line",row='age',col='sex')
plt.savefig('Q5a.png',dpi=300,bbox_inches='tight')
plt.show()
```


age = 15-24 years | sex = female
age = 15-24 years | sex = male
age = 25-34 years | sex = female
age = 25-34 years | sex = male
age = 35-54 years | sex = female
age = 35-54 years | sex = male
age = 5-14 years | sex = female
age = 5-14 years | sex = male
age = 55-74 years | sex = female
age = 55-74 years | sex = male
age = 75+ years | sex = female
age = 75+ years | sex = male
suicides/100k pop
year
sex
female
male

[635]:
```
seaborn.relplot(data=new_df, x="year", y="suicides/100k␣
↪pop",hue="age",kind="line")
plt.savefig('Q5b.png',dpi=300,bbox_inches='tight')
plt.show()
```



[636]:
```
seaborn.relplot(data=new_df, x="year", y="suicides/100k pop",␣
↪hue="sex",kind="line")
plt.savefig('Q5c.png',dpi=300,bbox_inches='tight')
plt.show()
```



0.5 Question 7

0.5.1 Add continent column to Suicides Rates Dataset

[13]:
```
continent_name = []

for i in range(len(suicide)):
    if("Korea" not in suicide.country[i] and "Grenadines" not in suicide.
 ↪country[i]):
        country_code = pc.country_name_to_country_alpha2(suicide.country[i],␣
 ↪cn_name_format="default")
        continent_name.append(pc.country_alpha2_to_continent_code(country_code))
    elif("Korea" in suicide.country[i]):
        continent_name.append('AS')
    elif("Grenadines" in suicide.country[i]):
        continent_name.append('NA')

suicide['Continent'] = continent_name
```

```
print("List of continents:")
print(set(continent_name))
print(suicide)
```

```
List of continents:
{'AS', 'EU', 'OC', 'SA', 'NA', 'AF'}
          country  year     sex          age  population   gdp_for_year ($)   \
0         Albania  1987    male  15-24 years      312900          2156624900
1         Albania  1987    male  35-54 years      308000          2156624900
2         Albania  1987  female  15-24 years      289700          2156624900
3         Albania  1987    male    75+ years       21800          2156624900
4         Albania  1987    male  25-34 years      274300          2156624900
…           …   …     …          …         …                  …
27815  Uzbekistan  2014  female  35-54 years     3620833         63067077179
27816  Uzbekistan  2014  female    75+ years      348465         63067077179
27817  Uzbekistan  2014    male   5-14 years     2762158         63067077179
27818  Uzbekistan  2014  female   5-14 years     2631600         63067077179
27819  Uzbekistan  2014  female  55-74 years     1438935         63067077179

       gdp_per_capita ($)       generation  suicides_no  suicides/100k pop  \
0                     796     Generation X           21               6.71
1                     796           Silent           16               5.19
2                     796     Generation X           14               4.83
3                     796  G.I. Generation            1               4.59
4                     796          Boomers            9               3.28
…                   …              …          …                …
27815                2309     Generation X          107               2.96
27816                2309           Silent            9               2.58
27817                2309     Generation Z           60               2.17
27818                2309     Generation Z           44               1.67
27819                2309          Boomers           21               1.46

      Continent
0            EU
1            EU
2            EU
3            EU
4            EU
…          …
27815        AS
27816        AS
27817        AS
27818        AS
27819        AS

[27820 rows x 11 columns]
```

0.5.2 Drop unused target variables and convert categorical variables to one-hot encoding

0.5.3 Bike Sharing Dataset

[14]:
```
bike_LR = bike.drop(['dteday','casual','registered'], axis=1)
bike_LR = pd.get_dummies(bike_LR,␣
 ↪columns=['season','mnth','weekday','weathersit'], drop_first=False)
print('Dataframe after one-hot encoding categorical variables:')
print(bike_LR)
```

```
Dataframe after one-hot encoding categorical variables:
     yr  holiday  workingday      temp     atemp       hum  windspeed   cnt  \
0     0        0           0  0.344167  0.363625  0.805833   0.160446   985
1     0        0           0  0.363478  0.353739  0.696087   0.248539   801
2     0        0           1  0.196364  0.189405  0.437273   0.248309  1349
3     0        0           1  0.200000  0.212122  0.590435   0.160296  1562
4     0        0           1  0.226957  0.229270  0.436957   0.186900  1600
..   ..      ...         ...       ...       ...       ...        ...   ...
726   1        0           1  0.254167  0.226642  0.652917   0.350133  2114
727   1        0           1  0.253333  0.255046  0.590000   0.155471  3095
728   1        0           0  0.253333  0.242400  0.752917   0.124383  1341
729   1        0           0  0.255833  0.231700  0.483333   0.350754  1796
730   1        0           1  0.215833  0.223487  0.577500   0.154846  2729

     season_1  season_2  season_3  season_4  mnth_1  mnth_2  mnth_3  mnth_4  \
0           1         0         0         0       1       0       0       0
1           1         0         0         0       1       0       0       0
2           1         0         0         0       1       0       0       0
3           1         0         0         0       1       0       0       0
4           1         0         0         0       1       0       0       0
..        ...       ...       ...       ...     ...     ...     ...     ...
726         1         0         0         0       0       0       0       0
727         1         0         0         0       0       0       0       0
728         1         0         0         0       0       0       0       0
729         1         0         0         0       0       0       0       0
730         1         0         0         0       0       0       0       0

     mnth_5  mnth_6  mnth_7  mnth_8  mnth_9  mnth_10  mnth_11  mnth_12  \
0         0       0       0       0       0        0        0        0
1         0       0       0       0       0        0        0        0
2         0       0       0       0       0        0        0        0
3         0       0       0       0       0        0        0        0
4         0       0       0       0       0        0        0        0
..      ...     ...     ...     ...     ...      ...      ...      ...
726       0       0       0       0       0        0        0        1
727       0       0       0       0       0        0        0        1
728       0       0       0       0       0        0        0        1
```

```
729       0       0       0       0       0       0       0       1
730       0       0       0       0       0       0       0       1

     weekday_0  weekday_1  weekday_2  weekday_3  weekday_4  weekday_5  \
0            0          0          0          0          0          0
1            1          0          0          0          0          0
2            0          1          0          0          0          0
3            0          0          1          0          0          0
4            0          0          0          1          0          0
..         …          …          …          …          …          …
726          0          0          0          0          1          0
727          0          0          0          0          0          1
728          0          0          0          0          0          0
729          1          0          0          0          0          0
730          0          1          0          0          0          0

     weekday_6  weathersit_1  weathersit_2  weathersit_3
0            1             0             1             0
1            0             0             1             0
2            0             1             0             0
3            0             1             0             0
4            0             1             0             0
..         …             …             …             …
726          0             0             1             0
727          0             0             1             0
728          1             0             1             0
729          0             1             0             0
730          0             0             1             0

[731 rows x 34 columns]
```

0.5.4 Suicide Rates Dataset

[15]:
```
suicide_LR = suicide.drop(['country','suicides_no'], axis=1)
suicide_LR = pd.get_dummies(suicide_LR,␣
 ↪columns=['sex','age','generation','Continent'], drop_first=False)
print('Dataframe after one-hot encoding categorical variables:')
print(suicide_LR)
```

```
Dataframe after one-hot encoding categorical variables:
       year  population   gdp_for_year ($)   gdp_per_capita ($)  \
0      1987      312900         2156624900                  796
1      1987      308000         2156624900                  796
2      1987      289700         2156624900                  796
3      1987       21800         2156624900                  796
4      1987      274300         2156624900                  796
…     …          …                 …                 …
27815  2014     3620833        63067077179                2309
```

```
27816  2014      348465         63067077179               2309
27817  2014     2762158         63067077179               2309
27818  2014     2631600         63067077179               2309
27819  2014     1438935         63067077179               2309

       suicides/100k pop  sex_female  sex_male  age_15-24 years  \
0                   6.71           0         1                1
1                   5.19           0         1                0
2                   4.83           1         0                1
3                   4.59           0         1                0
4                   3.28           0         1                0
…                   …          …       …              …
27815               2.96           1         0                0
27816               2.58           1         0                0
27817               2.17           0         1                0
27818               1.67           1         0                0
27819               1.46           1         0                0

       age_25-34 years  age_35-54 years  age_5-14 years  age_55-74 years  \
0                    0                0               0                0
1                    0                1               0                0
2                    0                0               0                0
3                    0                0               0                0
4                    1                0               0                0
…                …               …             …               …
27815                0                1               0                0
27816                0                0               0                0
27817                0                0               1                0
27818                0                0               1                0
27819                0                0               0                1

       age_75+ years  generation_Boomers  generation_G.I. Generation  \
0                  0                   0                           0
1                  0                   0                           0
2                  0                   0                           0
3                  1                   0                           1
4                  0                   1                           0
…                …                   …                         …
27815              0                   0                           0
27816              1                   0                           0
27817              0                   0                           0
27818              0                   0                           0
27819              0                   1                           0

       generation_Generation X  generation_Generation Z  \
0                            1                        0
1                            0                        0
2                            1                        0
```

```
3                            0                        0
4                            0                        0
…                          …                        …
27815                        1                        0
27816                        0                        0
27817                        0                        1
27818                        0                        1
27819                        0                        0

       generation_Millenials  generation_Silent  Continent_AF  Continent_AS  \
0                          0                  0             0             0
1                          0                  1             0             0
2                          0                  0             0             0
3                          0                  0             0             0
4                          0                  0             0             0
…                        …                  …             …             …
27815                      0                  0             0             1
27816                      0                  1             0             1
27817                      0                  0             0             1
27818                      0                  0             0             1
27819                      0                  0             0             1

       Continent_EU  Continent_NA  Continent_OC  Continent_SA
0                 1             0             0             0
1                 1             0             0             0
2                 1             0             0             0
3                 1             0             0             0
4                 1             0             0             0
…               …             …             …             …
27815             0             0             0             0
27816             0             0             0             0
27817             0             0             0             0
27818             0             0             0             0
27819             0             0             0             0

[27820 rows x 25 columns]
```

0.5.5 Video Transcoding Dataset

[16]:
```
transcode_LR = transcode_meas.drop(['b_size','umem'], axis=1)
transcode_LR = pd.get_dummies(transcode_LR, columns=['codec','o_codec'],␣
 ↪drop_first=False)
print('Dataframe after one-hot encoding categorical variables:')
print(transcode_LR)
```

```
Dataframe after one-hot encoding categorical variables:
        duration  width  height  bitrate  framerate    i      p  b  frames  \
0      130.35667    176     144    54590  12.000000   27   1537  0    1564
```

```
1      130.35667    176     144    54590  12.000000   27   1537  0    1564
2      130.35667    176     144    54590  12.000000   27   1537  0    1564
3      130.35667    176     144    54590  12.000000   27   1537  0    1564
4      130.35667    176     144    54590  12.000000   27   1537  0    1564
…          …    …     …       …        …  …    … ..     …
68779  972.27100    480     360   278822  29.000000  560  28580  0   29140
68780  129.88100    640     480   639331  30.162790   36   3855  0    3891
68781  249.68000    320     240   359345  25.068274  129   6113  0    6242
68782  183.62334   1280     720  2847539  29.000000   98   5405  0    5503
68783  294.61334    176     144    55242  12.000000   61   3474  0    3535

        i_size    p_size      size  o_bitrate  o_framerate  o_width  o_height  \
0        64483    825054    889537      56000        12.00      176       144
1        64483    825054    889537      56000        12.00      320       240
2        64483    825054    889537      56000        12.00      480       360
3        64483    825054    889537      56000        12.00      640       480
4        64483    825054    889537      56000        12.00     1280       720
…          …       …        …          …           …       …       …
68779  7324628  26561730  33886358     242000        24.00      640       480
68780   875784   9503846  10379630     539000        29.97     1920      1080
68781  1758664   9456514  11215178     539000        12.00      176       144
68782  5246294  60113035  65359329     539000        12.00      320       240
68783    84002   1950409   2034411     820000        24.00      176       144

        utime  codec_flv  codec_h264  codec_mpeg4  codec_vp8  o_codec_flv  \
0       0.612          0           0            1          0            0
1       0.980          0           0            1          0            0
2       1.216          0           0            1          0            0
3       1.692          0           0            1          0            0
4       3.456          0           0            1          0            0
…         …        …          …           …        …           …
68779   1.552          0           1            0          0            1
68780  18.557          0           0            0          1            0
68781   0.752          0           0            0          1            1
68782   5.444          0           1            0          0            0
68783   3.076          0           0            1          0            0

       o_codec_h264  o_codec_mpeg4  o_codec_vp8
0                 0              1            0
1                 0              1            0
2                 0              1            0
3                 0              1            0
4                 0              1            0
…               …              …          …
68779             0              0            0
68780             0              1            0
68781             0              0            0
68782             0              1            0
```

```
68783              1              0              0

[68784 rows x 25 columns]
```

0.6 Question 8

0.6.1 Extract training variables and target values

[17]:
```
XBike = bike_LR.loc[:, bike_LR.columns != 'cnt'].to_numpy()
YBike = bike_LR.cnt
XSuicide = suicide_LR.loc[:, suicide_LR.columns != 'suicides/100k pop'].
↪to_numpy()
YSuicide = suicide_LR["suicides/100k pop"]
XTranscode = transcode_LR.loc[:,  transcode_LR.columns != 'utime'].to_numpy()
YTranscode = transcode_LR["utime"]
```

0.6.2 Scaling

[18]:
```
bike_scale = StandardScaler()
XBike_S = bike_scale.fit_transform(XBike)
suicide_scale = StandardScaler()
XSuicide_S = suicide_scale.fit_transform(XSuicide)
transcode_scale = StandardScaler()
XTranscode_S = transcode_scale.fit_transform(XTranscode)
```

0.7 Question 9

0.7.1 Testing Feature Selection with Default Hyperparameters for Linear, Ridge and Lasso Regression

[19]:
```
bike_RMSE_MIR = []
bike_RMSE_FR = []
Suicide_RMSE_FR = []
Transcode_RMSE_FR = []

bike_RMSE_MIR_RR = []
bike_RMSE_FR_RR = []
Suicide_RMSE_FR_RR = []
Transcode_RMSE_FR_RR = []

bike_RMSE_MIR_LR = []
bike_RMSE_FR_LR = []
Suicide_RMSE_FR_LR = []
Transcode_RMSE_FR_LR = []


for i in range(1,XBike.shape[1]):
```

```
    print('Testing LR, bike dataset for k = ', i)
    XBikeCur_M = SelectKBest(score_func=mutual_info_regression, k=i).
↪fit_transform(XBike, YBike)
    XBikeCur_F = SelectKBest(score_func=f_regression, k=i).fit_transform(XBike,␣
↪YBike)

    BikeOut = cross_validate(LinearRegression(), XBikeCur_M, YBike,␣
↪scoring=['neg_root_mean_squared_error'], cv=10,n_jobs=-1)
    bike_RMSE_MIR.append(BikeOut['test_neg_root_mean_squared_error'].mean())
    BikeOut = cross_validate(LinearRegression(), XBikeCur_F, YBike,␣
↪scoring=['neg_root_mean_squared_error'], cv=10,n_jobs=-1)
    bike_RMSE_FR.append(BikeOut['test_neg_root_mean_squared_error'].mean())

    print('Testing RR, bike dataset for k = ', i)
    BikeOut = cross_validate(Ridge(), XBikeCur_M, YBike,␣
↪scoring=['neg_root_mean_squared_error'], cv=10,n_jobs=-1)
    bike_RMSE_MIR_RR.append(BikeOut['test_neg_root_mean_squared_error'].mean())
    BikeOut = cross_validate(Ridge(), XBikeCur_F, YBike,␣
↪scoring=['neg_root_mean_squared_error'], cv=10,n_jobs=-1)
    bike_RMSE_FR_RR.append(BikeOut['test_neg_root_mean_squared_error'].mean())

    print('Testing LaR, bike dataset for k = ', i)
    BikeOut = cross_validate(Lasso(), XBikeCur_M, YBike,␣
↪scoring=['neg_root_mean_squared_error'], cv=10,n_jobs=-1)
    bike_RMSE_MIR_LR.append(BikeOut['test_neg_root_mean_squared_error'].mean())
    BikeOut = cross_validate(Lasso(), XBikeCur_F, YBike,␣
↪scoring=['neg_root_mean_squared_error'], cv=10,n_jobs=-1)
    bike_RMSE_FR_LR.append(BikeOut['test_neg_root_mean_squared_error'].mean())
```

```
Testing LR, bike dataset for k =  1
Testing RR, bike dataset for k =  1
Testing LaR, bike dataset for k =  1
Testing LR, bike dataset for k =  2
Testing RR, bike dataset for k =  2
Testing LaR, bike dataset for k =  2
Testing LR, bike dataset for k =  3
Testing RR, bike dataset for k =  3
Testing LaR, bike dataset for k =  3
Testing LR, bike dataset for k =  4
Testing RR, bike dataset for k =  4
Testing LaR, bike dataset for k =  4
Testing LR, bike dataset for k =  5
Testing RR, bike dataset for k =  5
Testing LaR, bike dataset for k =  5
Testing LR, bike dataset for k =  6
Testing RR, bike dataset for k =  6
Testing LaR, bike dataset for k =  6
```

```
Testing LR, bike dataset for k =  7
Testing RR, bike dataset for k =  7
Testing LaR, bike dataset for k =  7
Testing LR, bike dataset for k =  8
Testing RR, bike dataset for k =  8
Testing LaR, bike dataset for k =  8
Testing LR, bike dataset for k =  9
Testing RR, bike dataset for k =  9
Testing LaR, bike dataset for k =  9
Testing LR, bike dataset for k =  10
Testing RR, bike dataset for k =  10
Testing LaR, bike dataset for k =  10
Testing LR, bike dataset for k =  11
Testing RR, bike dataset for k =  11
Testing LaR, bike dataset for k =  11
Testing LR, bike dataset for k =  12
Testing RR, bike dataset for k =  12
Testing LaR, bike dataset for k =  12
Testing LR, bike dataset for k =  13
Testing RR, bike dataset for k =  13
Testing LaR, bike dataset for k =  13
Testing LR, bike dataset for k =  14
Testing RR, bike dataset for k =  14
Testing LaR, bike dataset for k =  14
Testing LR, bike dataset for k =  15
Testing RR, bike dataset for k =  15
Testing LaR, bike dataset for k =  15
Testing LR, bike dataset for k =  16
Testing RR, bike dataset for k =  16
Testing LaR, bike dataset for k =  16
Testing LR, bike dataset for k =  17
Testing RR, bike dataset for k =  17
Testing LaR, bike dataset for k =  17
Testing LR, bike dataset for k =  18
Testing RR, bike dataset for k =  18
Testing LaR, bike dataset for k =  18
Testing LR, bike dataset for k =  19
Testing RR, bike dataset for k =  19
Testing LaR, bike dataset for k =  19
Testing LR, bike dataset for k =  20
Testing RR, bike dataset for k =  20
Testing LaR, bike dataset for k =  20
Testing LR, bike dataset for k =  21
Testing RR, bike dataset for k =  21
Testing LaR, bike dataset for k =  21
Testing LR, bike dataset for k =  22
Testing RR, bike dataset for k =  22
Testing LaR, bike dataset for k =  22
```

```
Testing LR, bike dataset for k =  23
Testing RR, bike dataset for k =  23
Testing LaR, bike dataset for k =  23
Testing LR, bike dataset for k =  24
Testing RR, bike dataset for k =  24
Testing LaR, bike dataset for k =  24
Testing LR, bike dataset for k =  25
Testing RR, bike dataset for k =  25
Testing LaR, bike dataset for k =  25
Testing LR, bike dataset for k =  26
Testing RR, bike dataset for k =  26
Testing LaR, bike dataset for k =  26
Testing LR, bike dataset for k =  27
Testing RR, bike dataset for k =  27
Testing LaR, bike dataset for k =  27
Testing LR, bike dataset for k =  28
Testing RR, bike dataset for k =  28
Testing LaR, bike dataset for k =  28
Testing LR, bike dataset for k =  29
Testing RR, bike dataset for k =  29
Testing LaR, bike dataset for k =  29
Testing LR, bike dataset for k =  30
Testing RR, bike dataset for k =  30
Testing LaR, bike dataset for k =  30
Testing LR, bike dataset for k =  31
Testing RR, bike dataset for k =  31
Testing LaR, bike dataset for k =  31
Testing LR, bike dataset for k =  32
Testing RR, bike dataset for k =  32
Testing LaR, bike dataset for k =  32
```

[20]:
```
for i in range(1,XSuicide.shape[1]):
    print('Testing LR, Suicide dataset for k = ', i)
    XSuicideCur_F = SelectKBest(score_func=f_regression, k=i).
↪fit_transform(XSuicide, YSuicide)

    SuicideOut = cross_validate(LinearRegression(), XSuicideCur_F, YSuicide,␣
↪scoring=['neg_root_mean_squared_error'], cv=10,n_jobs=-1)
    Suicide_RMSE_FR.append(SuicideOut['test_neg_root_mean_squared_error'].
↪mean())

    print('Testing RR, Suicide dataset for k = ', i)
    SuicideOut = cross_validate(Ridge(), XSuicideCur_F, YSuicide,␣
↪scoring=['neg_root_mean_squared_error'], cv=10,n_jobs=-1)
    Suicide_RMSE_FR_RR.append(SuicideOut['test_neg_root_mean_squared_error'].
↪mean())
```

```
    print('Testing LaR, Suicide dataset for k = ', i)
    SuicideOut = cross_validate(Lasso(), XSuicideCur_F, YSuicide,␣
↪scoring=['neg_root_mean_squared_error'], cv=10,n_jobs=-1)
    Suicide_RMSE_FR_LR.append(SuicideOut['test_neg_root_mean_squared_error'].
↪mean())
```

```
Testing LR, Suicide dataset for k =  1
Testing RR, Suicide dataset for k =  1
Testing LaR, Suicide dataset for k =  1
Testing LR, Suicide dataset for k =  2
Testing RR, Suicide dataset for k =  2
Testing LaR, Suicide dataset for k =  2
Testing LR, Suicide dataset for k =  3
Testing RR, Suicide dataset for k =  3
Testing LaR, Suicide dataset for k =  3
Testing LR, Suicide dataset for k =  4
Testing RR, Suicide dataset for k =  4
Testing LaR, Suicide dataset for k =  4
Testing LR, Suicide dataset for k =  5
Testing RR, Suicide dataset for k =  5
Testing LaR, Suicide dataset for k =  5
Testing LR, Suicide dataset for k =  6
Testing RR, Suicide dataset for k =  6
Testing LaR, Suicide dataset for k =  6
Testing LR, Suicide dataset for k =  7
Testing RR, Suicide dataset for k =  7
Testing LaR, Suicide dataset for k =  7
Testing LR, Suicide dataset for k =  8
Testing RR, Suicide dataset for k =  8
Testing LaR, Suicide dataset for k =  8
Testing LR, Suicide dataset for k =  9
Testing RR, Suicide dataset for k =  9
Testing LaR, Suicide dataset for k =  9
Testing LR, Suicide dataset for k =  10
Testing RR, Suicide dataset for k =  10
Testing LaR, Suicide dataset for k =  10
Testing LR, Suicide dataset for k =  11
Testing RR, Suicide dataset for k =  11
Testing LaR, Suicide dataset for k =  11
Testing LR, Suicide dataset for k =  12
Testing RR, Suicide dataset for k =  12
Testing LaR, Suicide dataset for k =  12
Testing LR, Suicide dataset for k =  13
Testing RR, Suicide dataset for k =  13
Testing LaR, Suicide dataset for k =  13
Testing LR, Suicide dataset for k =  14
Testing RR, Suicide dataset for k =  14
```

```
Testing LaR, Suicide dataset for k =  14
Testing LR, Suicide dataset for k =  15
Testing RR, Suicide dataset for k =  15
Testing LaR, Suicide dataset for k =  15
Testing LR, Suicide dataset for k =  16
Testing RR, Suicide dataset for k =  16
Testing LaR, Suicide dataset for k =  16
Testing LR, Suicide dataset for k =  17
Testing RR, Suicide dataset for k =  17
Testing LaR, Suicide dataset for k =  17
Testing LR, Suicide dataset for k =  18
Testing RR, Suicide dataset for k =  18
Testing LaR, Suicide dataset for k =  18
Testing LR, Suicide dataset for k =  19
Testing RR, Suicide dataset for k =  19
Testing LaR, Suicide dataset for k =  19
Testing LR, Suicide dataset for k =  20
Testing RR, Suicide dataset for k =  20
Testing LaR, Suicide dataset for k =  20
Testing LR, Suicide dataset for k =  21
Testing RR, Suicide dataset for k =  21
Testing LaR, Suicide dataset for k =  21
Testing LR, Suicide dataset for k =  22
Testing RR, Suicide dataset for k =  22
Testing LaR, Suicide dataset for k =  22
Testing LR, Suicide dataset for k =  23
Testing RR, Suicide dataset for k =  23
Testing LaR, Suicide dataset for k =  23
```

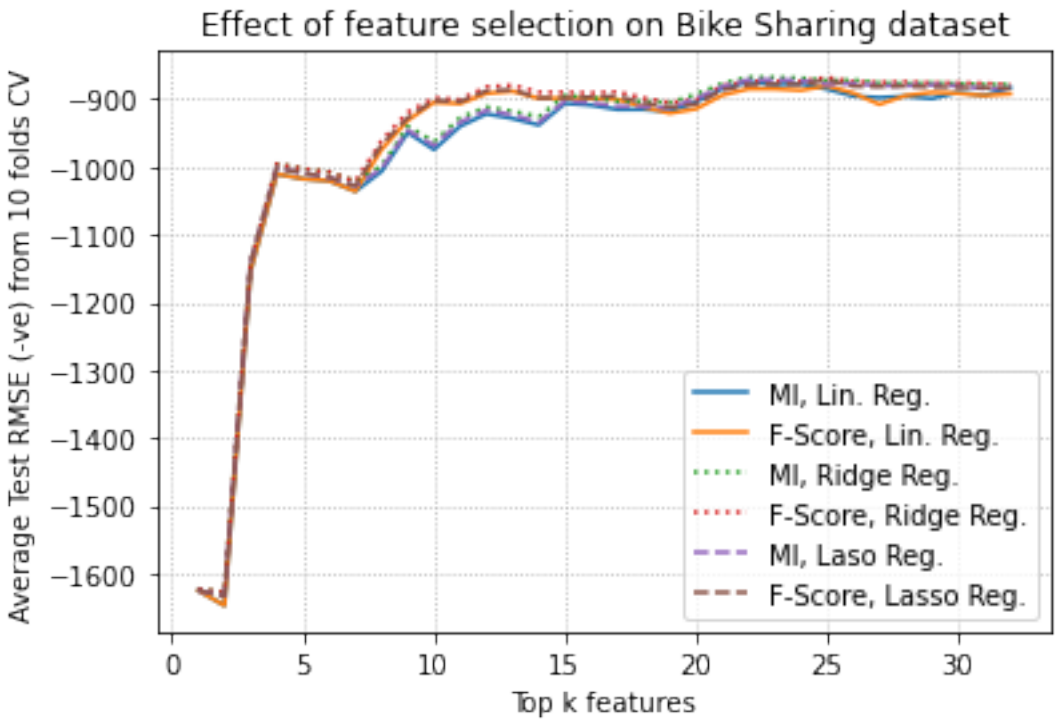
[21]:
```
for i in range(1,XTranscode.shape[1]):
    print('Testing LR, Transcode dataset for k = ', i)
    XTranscodeCur_F = SelectKBest(score_func=f_regression, k=i).
↪fit_transform(XTranscode, YTranscode)

    TranscodeOut = cross_validate(LinearRegression(), XTranscodeCur_F,␣
↪YTranscode, scoring=['neg_root_mean_squared_error'], cv=10,n_jobs=-1)
    Transcode_RMSE_FR.append(TranscodeOut['test_neg_root_mean_squared_error'].
↪mean())

    print('Testing RR, Transcode dataset for k = ', i)
    TranscodeOut = cross_validate(Ridge(), XTranscodeCur_F, YTranscode,␣
↪scoring=['neg_root_mean_squared_error'], cv=10,n_jobs=-1)
    Transcode_RMSE_FR_RR.
↪append(TranscodeOut['test_neg_root_mean_squared_error'].mean())

    print('Testing LaR, Transcode dataset for k = ', i)
```

```
    TranscodeOut = cross_validate(Lasso(), XTranscodeCur_F, YTranscode,␣
↪scoring=['neg_root_mean_squared_error'], cv=10,n_jobs=-1)
    Transcode_RMSE_FR_LR.
↪append(TranscodeOut['test_neg_root_mean_squared_error'].mean())
```

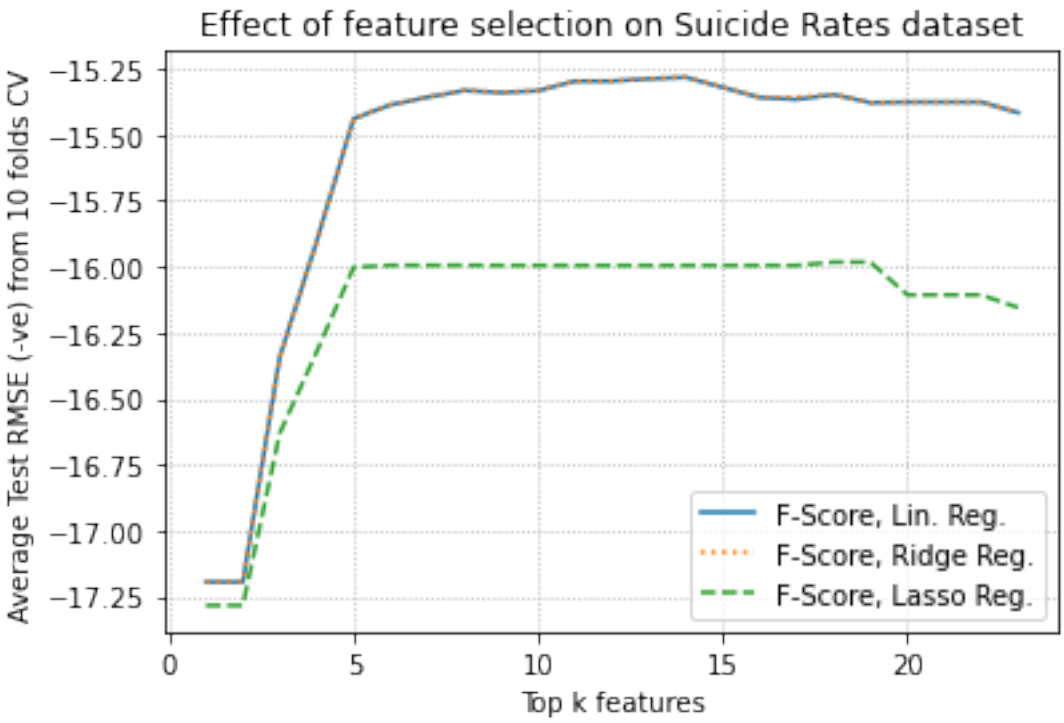
```
Testing LR, Transcode dataset for k =  1
Testing RR, Transcode dataset for k =  1
Testing LaR, Transcode dataset for k =  1
Testing LR, Transcode dataset for k =  2
Testing RR, Transcode dataset for k =  2
Testing LaR, Transcode dataset for k =  2
Testing LR, Transcode dataset for k =  3
Testing RR, Transcode dataset for k =  3
Testing LaR, Transcode dataset for k =  3
Testing LR, Transcode dataset for k =  4
Testing RR, Transcode dataset for k =  4
Testing LaR, Transcode dataset for k =  4
Testing LR, Transcode dataset for k =  5
Testing RR, Transcode dataset for k =  5
Testing LaR, Transcode dataset for k =  5
Testing LR, Transcode dataset for k =  6
Testing RR, Transcode dataset for k =  6
Testing LaR, Transcode dataset for k =  6
Testing LR, Transcode dataset for k =  7
Testing RR, Transcode dataset for k =  7
Testing LaR, Transcode dataset for k =  7
Testing LR, Transcode dataset for k =  8
Testing RR, Transcode dataset for k =  8
Testing LaR, Transcode dataset for k =  8
Testing LR, Transcode dataset for k =  9
Testing RR, Transcode dataset for k =  9
Testing LaR, Transcode dataset for k =  9
Testing LR, Transcode dataset for k =  10
Testing RR, Transcode dataset for k =  10
Testing LaR, Transcode dataset for k =  10
Testing LR, Transcode dataset for k =  11
Testing RR, Transcode dataset for k =  11
Testing LaR, Transcode dataset for k =  11
Testing LR, Transcode dataset for k =  12
Testing RR, Transcode dataset for k =  12
Testing LaR, Transcode dataset for k =  12
Testing LR, Transcode dataset for k =  13
Testing RR, Transcode dataset for k =  13
Testing LaR, Transcode dataset for k =  13
Testing LR, Transcode dataset for k =  14
Testing RR, Transcode dataset for k =  14
Testing LaR, Transcode dataset for k =  14
```

```
Testing LR, Transcode dataset for k =  15
Testing RR, Transcode dataset for k =  15
Testing LaR, Transcode dataset for k =  15
Testing LR, Transcode dataset for k =  16
Testing RR, Transcode dataset for k =  16
Testing LaR, Transcode dataset for k =  16
Testing LR, Transcode dataset for k =  17
Testing RR, Transcode dataset for k =  17
Testing LaR, Transcode dataset for k =  17
Testing LR, Transcode dataset for k =  18
Testing RR, Transcode dataset for k =  18
Testing LaR, Transcode dataset for k =  18
Testing LR, Transcode dataset for k =  19
Testing RR, Transcode dataset for k =  19
Testing LaR, Transcode dataset for k =  19
Testing LR, Transcode dataset for k =  20
Testing RR, Transcode dataset for k =  20
Testing LaR, Transcode dataset for k =  20
Testing LR, Transcode dataset for k =  21
Testing RR, Transcode dataset for k =  21
Testing LaR, Transcode dataset for k =  21
Testing LR, Transcode dataset for k =  22
Testing RR, Transcode dataset for k =  22
Testing LaR, Transcode dataset for k =  22
Testing LR, Transcode dataset for k =  23
Testing RR, Transcode dataset for k =  23
Testing LaR, Transcode dataset for k =  23
```

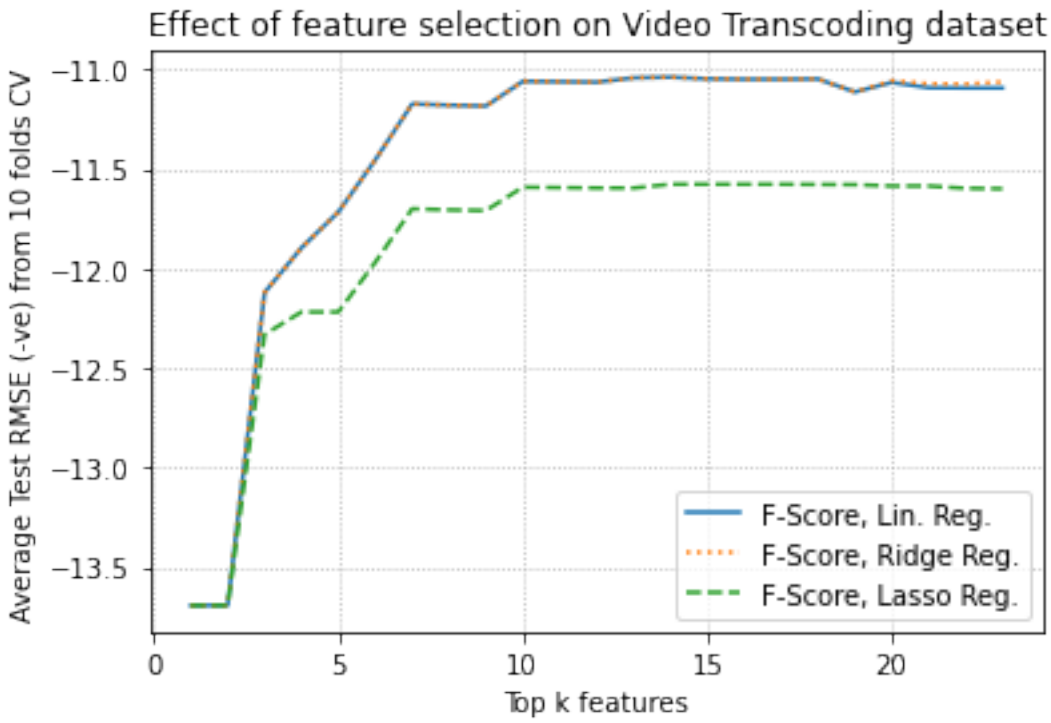
[22]:
```
plt.plot(np.arange(1,len(bike_RMSE_MIR)+1,1),bike_RMSE_MIR)
plt.plot(np.arange(1,len(bike_RMSE_FR)+1,1),bike_RMSE_FR)
plt.plot(np.arange(1,len(bike_RMSE_MIR_RR)+1,1),bike_RMSE_MIR_RR,':')
plt.plot(np.arange(1,len(bike_RMSE_FR_RR)+1,1),bike_RMSE_FR_RR,':')
plt.plot(np.arange(1,len(bike_RMSE_MIR_LR)+1,1),bike_RMSE_MIR_LR,'--')
plt.plot(np.arange(1,len(bike_RMSE_FR_LR)+1,1),bike_RMSE_FR_LR,'--')
plt.legend(['MI, Lin. Reg.','F-Score, Lin. Reg.','MI, Ridge Reg.',
            'F-Score, Ridge Reg.','MI, Laso Reg.','F-Score, Lasso Reg.
 ↪'],loc='best')
plt.grid(linestyle=':')
plt.xlabel('Top k features')
plt.ylabel('Average Test RMSE (-ve) from 10 folds CV')
plt.title('Effect of feature selection on Bike Sharing dataset')
plt.savefig('Q9a.png',dpi=300,bbox_inches='tight')
plt.show()
```



[23]:
```
plt.plot(np.arange(1,len(Suicide_RMSE_FR)+1,1),Suicide_RMSE_FR)
plt.plot(np.arange(1,len(Suicide_RMSE_FR_RR)+1,1),Suicide_RMSE_FR_RR,':')
plt.plot(np.arange(1,len(Suicide_RMSE_FR_LR)+1,1),Suicide_RMSE_FR_LR,'--')
plt.legend(['F-Score, Lin. Reg.','F-Score, Ridge Reg.','F-Score, Lasso Reg.
 ↪'],loc='best')
plt.grid(linestyle=':')
plt.xlabel('Top k features')
plt.ylabel('Average Test RMSE (-ve) from 10 folds CV')
plt.title('Effect of feature selection on Suicide Rates dataset')
plt.savefig('Q9b.png',dpi=300,bbox_inches='tight')
plt.show()
```



[24]:
```
plt.plot(np.arange(1,len(Transcode_RMSE_FR)+1,1),Transcode_RMSE_FR)
plt.plot(np.arange(1,len(Transcode_RMSE_FR_RR)+1,1),Transcode_RMSE_FR_RR,':')
plt.plot(np.arange(1,len(Transcode_RMSE_FR_LR)+1,1),Transcode_RMSE_FR_LR,'--')
plt.legend(['F-Score, Lin. Reg.','F-Score, Ridge Reg.','F-Score, Lasso Reg.
 ↪'],loc='best')
plt.grid(linestyle=':')
plt.xlabel('Top k features')
plt.ylabel('Average Test RMSE (-ve) from 10 folds CV')
plt.title('Effect of feature selection on Video Transcoding dataset')
plt.savefig('Q9c.png',dpi=300,bbox_inches='tight')
plt.show()
```



0.8 Question 10 to 13

[25]:
```
k_val = 10

XBikeCur_F = SelectKBest(score_func=f_regression, k=k_val).fit_transform(XBike,␣
↪YBike)
XBikeCur_MIR = SelectKBest(score_func=mutual_info_regression, k=k_val).
↪fit_transform(XBike, YBike)
XBikeCur_FS = SelectKBest(score_func=f_regression, k=k_val).
↪fit_transform(XBike_S, YBike)
XBikeCur_MIRS = SelectKBest(score_func=mutual_info_regression, k=k_val).
↪fit_transform(XBike_S, YBike)

XSuicideCur_F = SelectKBest(score_func=f_regression, k=k_val).
↪fit_transform(XSuicide, YSuicide)
XSuicideCur_FS = SelectKBest(score_func=f_regression, k=k_val).
↪fit_transform(XSuicide_S, YSuicide)

XTranscodeCur_F = SelectKBest(score_func=f_regression, k=k_val).
↪fit_transform(XTranscode, YTranscode)
XTranscodeCur_FS = SelectKBest(score_func=f_regression, k=k_val).
↪fit_transform(XTranscode_S, YTranscode)
```

0.8.1 Linear Regression

[38]:
```
BikeOut = cross_validate(LinearRegression(), XBikeCur_F, YBike,␣
↪scoring=['neg_root_mean_squared_error'],␣
↪cv=10,n_jobs=-1,return_train_score=True)
print('• No standardization, bike dataset, F1, linear regression:␣
↪Test=',BikeOut['test_neg_root_mean_squared_error'].
↪mean(),',Train=',BikeOut['train_neg_root_mean_squared_error'].mean())
BikeOut = cross_validate(LinearRegression(), XBikeCur_MIR, YBike,␣
↪scoring=['neg_root_mean_squared_error'],␣
↪cv=10,n_jobs=-1,return_train_score=True)
print('• No standardization, bike dataset, MI, linear regression:␣
↪Test=',BikeOut['test_neg_root_mean_squared_error'].
↪mean(),',Train=',BikeOut['train_neg_root_mean_squared_error'].mean())
BikeOut = cross_validate(LinearRegression(), XBikeCur_FS, YBike,␣
↪scoring=['neg_root_mean_squared_error'],␣
↪cv=10,n_jobs=-1,return_train_score=True)
print('• Standardization, bike dataset, F1, linear regression: ␣
↪Test=',BikeOut['test_neg_root_mean_squared_error'].
↪mean(),',Train=',BikeOut['train_neg_root_mean_squared_error'].mean())
BikeOut = cross_validate(LinearRegression(), XBikeCur_MIRS, YBike,␣
↪scoring=['neg_root_mean_squared_error'],␣
↪cv=10,n_jobs=-1,return_train_score=True)
print('• Standardization, bike dataset, MI, linear regression:␣
↪Test=',BikeOut['test_neg_root_mean_squared_error'].
↪mean(),',Train=',BikeOut['train_neg_root_mean_squared_error'].mean())

SuicideOut = cross_validate(LinearRegression(), XSuicideCur_F, YSuicide,␣
↪scoring=['neg_root_mean_squared_error'],␣
↪cv=10,n_jobs=-1,return_train_score=True)
print('• No standardization, Suicide dataset, F1, linear regression:␣
↪Test=',SuicideOut['test_neg_root_mean_squared_error'].
↪mean(),',Train=',SuicideOut['train_neg_root_mean_squared_error'].mean())
SuicideOut = cross_validate(LinearRegression(), XSuicideCur_FS, YSuicide,␣
↪scoring=['neg_root_mean_squared_error'],␣
↪cv=10,n_jobs=-1,return_train_score=True)
print('• Standardization, Suicide dataset, F1, linear regression:␣
↪Test=',SuicideOut['test_neg_root_mean_squared_error'].
↪mean(),',Train=',SuicideOut['train_neg_root_mean_squared_error'].mean())

TranscodeOut = cross_validate(LinearRegression(), XTranscodeCur_F, YTranscode,␣
↪scoring=['neg_root_mean_squared_error'],␣
↪cv=10,n_jobs=-1,return_train_score=True)
print('• No standardization, Transcode dataset, F1, linear regression:␣
↪Test=',TranscodeOut['test_neg_root_mean_squared_error'].
↪mean(),',Train=',TranscodeOut['train_neg_root_mean_squared_error'].mean())
```

```
TranscodeOut = cross_validate(LinearRegression(), XTranscodeCur_FS, YTranscode,␣
↪scoring=['neg_root_mean_squared_error'],␣
↪cv=10,n_jobs=-1,return_train_score=True)
print('• Standardization, Transcode dataset, F1, linear regression:␣
↪Test=',TranscodeOut['test_neg_root_mean_squared_error'].
↪mean(),',Train=',TranscodeOut['train_neg_root_mean_squared_error'].mean())
```

```
• No standardization, bike dataset, F1, linear regression: Test=
-904.875354514855 ,Train= -836.3621038950808
• No standardization, bike dataset, MI, linear regression: Test=
-974.4892680662463 ,Train= -871.9979472259711
• Standardization, bike dataset, F1, linear regression:  Test=
-904.8753545148545 ,Train= -836.3621038950808
• Standardization, bike dataset, MI, linear regression: Test= -974.4892680662466
,Train= -871.9979472259711
• No standardization, Suicide dataset, F1, linear regression: Test=
-15.333207286266429 ,Train= -15.42061968119477
• Standardization, Suicide dataset, F1, linear regression: Test=
-15.332744465450682 ,Train= -15.420589566680139
• No standardization, Transcode dataset, F1, linear regression: Test=
-11.059454228951019 ,Train= -11.038710923489962
• Standardization, Transcode dataset, F1, linear regression: Test=
-11.059454228951008 ,Train= -11.038710923489962
```

0.8.2 Ridge Regression

[39]:
```
pipe_RR = Pipeline([('model', Ridge(random_state=42))])
param_grid = {
    'model__alpha': [10.0**x for x in np.arange(-4,4)]
}
```

[40]:
```
print("Testing bike..\n")
gridBikeRR_F = GridSearchCV(pipe_RR, param_grid=param_grid, cv=10, n_jobs=-1,␣
↪verbose=1,
                    scoring='neg_root_mean_squared_error',␣
↪return_train_score=True).fit(XBikeCur_F, YBike)
gridBikeRR_FS = GridSearchCV(pipe_RR, param_grid=param_grid, cv=10, n_jobs=-1,␣
↪verbose=1,
                    scoring='neg_root_mean_squared_error',␣
↪return_train_score=True).fit(XBikeCur_FS, YBike)
gridBikeRR_M = GridSearchCV(pipe_RR, param_grid=param_grid, cv=10, n_jobs=-1,␣
↪verbose=1,
                    scoring='neg_root_mean_squared_error',␣
↪return_train_score=True).fit(XBikeCur_MIR, YBike)
gridBikeRR_MS = GridSearchCV(pipe_RR, param_grid=param_grid, cv=10, n_jobs=-1,␣
↪verbose=1,
```

```
                     scoring='neg_root_mean_squared_error',␣
 ↪return_train_score=True).fit(XBikeCur_MIRS, YBike)
print("Testing suicide..\n")
gridSuicideRR_F = GridSearchCV(pipe_RR, param_grid=param_grid, cv=10,␣
 ↪n_jobs=-1, verbose=1,
                     scoring='neg_root_mean_squared_error',␣
 ↪return_train_score=True).fit(XSuicideCur_F, YSuicide)
gridSuicideRR_FS = GridSearchCV(pipe_RR, param_grid=param_grid, cv=10,␣
 ↪n_jobs=-1, verbose=1,
                     scoring='neg_root_mean_squared_error',␣
 ↪return_train_score=True).fit(XSuicideCur_FS, YSuicide)
print("Testing transcoding..\n")
gridTranscodeRR_F = GridSearchCV(pipe_RR, param_grid=param_grid, cv=10,␣
 ↪n_jobs=-1, verbose=1,
                     scoring='neg_root_mean_squared_error',␣
 ↪return_train_score=True).fit(XTranscodeCur_F, YTranscode)
gridTranscodeRR_FS = GridSearchCV(pipe_RR, param_grid=param_grid, cv=10,␣
 ↪n_jobs=-1, verbose=1,
                     scoring='neg_root_mean_squared_error',␣
 ↪return_train_score=True).fit(XTranscodeCur_FS, YTranscode)
```

```
Testing bike..

Fitting 10 folds for each of 8 candidates, totalling 80 fits
Fitting 10 folds for each of 8 candidates, totalling 80 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    0.1s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    0.1s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    0.1s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    0.1s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
```

```
Fitting 10 folds for each of 8 candidates, totalling 80 fits
Fitting 10 folds for each of 8 candidates, totalling 80 fits
Testing suicide..

Fitting 10 folds for each of 8 candidates, totalling 80 fits
```

```
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    0.2s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
```

```
Fitting 10 folds for each of 8 candidates, totalling 80 fits
```

```
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    0.2s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
```

```
Testing transcoding..

Fitting 10 folds for each of 8 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    0.3s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 8 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    0.3s finished
```

[54]:
```
print('• No standardization, bike dataset, F1, ridge reg., Test RMSE:
 ↪',gridBikeRR_F.best_score_,
      ',alpha:',gridBikeRR_F.best_params_,'train RMSE',max(gridBikeRR_F.
 ↪cv_results_['mean_train_score']))
print('• Standardization, bike dataset, F1, ridge reg., Test RMSE:
 ↪',gridBikeRR_FS.best_score_,
      ',alpha:',gridBikeRR_FS.best_params_,'train RMSE',max(gridBikeRR_FS.
 ↪cv_results_['mean_train_score']))
print('• No standardization, bike dataset, MI, ridge reg., Test RMSE:
 ↪',gridBikeRR_M.best_score_,
      ',alpha:',gridBikeRR_F.best_params_,'train RMSE',max(gridBikeRR_M.
 ↪cv_results_['mean_train_score']))
print('• Standardization, bike dataset, MI, ridge reg., Test RMSE:
 ↪',gridBikeRR_MS.best_score_,
      ',alpha:',gridBikeRR_MS.best_params_,'train RMSE',max(gridBikeRR_MS.
 ↪cv_results_['mean_train_score']))
print('• No standardization, suicide dataset, F1, ridge reg., Test RMSE:
 ↪',gridSuicideRR_F.best_score_,
      ',alpha:',gridSuicideRR_F.best_params_,'train RMSE',max(gridSuicideRR_F.
 ↪cv_results_['mean_train_score']))
print('• Standardization, suicide dataset, F1, ridge reg., Test RMSE:
 ↪',gridSuicideRR_FS.best_score_,
      ',alpha:',gridSuicideRR_FS.best_params_,'train RMSE',max(gridSuicideRR_FS.
 ↪cv_results_['mean_train_score']))
print('• No standardization, transcoding dataset, F1, ridge reg., Test RMSE:
 ↪',gridTranscodeRR_F.best_score_,
      ',alpha:',gridTranscodeRR_F.best_params_,'train␣
 ↪RMSE',max(gridTranscodeRR_F.cv_results_['mean_train_score']))
print('• Standardization, transcoding dataset, F1, ridge reg., Test RMSE:
 ↪',gridTranscodeRR_FS.best_score_,
      ',alpha:',gridTranscodeRR_FS.best_params_,'train␣
 ↪RMSE',max(gridTranscodeRR_FS.cv_results_['mean_train_score']))
```

```
• No standardization, bike dataset, F1, ridge reg., Test RMSE: -892.70230942667
,alpha: {'model__alpha': 10.0} train RMSE -836.3621041845709
• Standardization, bike dataset, F1, ridge reg., Test RMSE: -878.1469958606837
,alpha: {'model__alpha': 100.0} train RMSE -836.3621038954203
```

• No standardization, bike dataset, MI, ridge reg., Test RMSE:
-962.7636703128888 ,alpha: {'model__alpha': 10.0} train RMSE -871.9979474460961
• Standardization, bike dataset, MI, ridge reg., Test RMSE: -935.7339113922999
,alpha: {'model__alpha': 100.0} train RMSE -871.9979472261564
• No standardization, suicide dataset, F1, ridge reg., Test RMSE:
-15.32570785324082 ,alpha: {'model__alpha': 1000.0} train RMSE
-15.42018215133271
• Standardization, suicide dataset, F1, ridge reg., Test RMSE:
-15.322322304708754 ,alpha: {'model__alpha': 1000.0} train RMSE
-15.42018215133271
• No standardization, transcoding dataset, F1, ridge reg., Test RMSE:
-11.059236787276069 ,alpha: {'model__alpha': 100.0} train RMSE
-11.038710923489962
• Standardization, transcoding dataset, F1, ridge reg., Test RMSE:
-11.059033334149994 ,alpha: {'model__alpha': 100.0} train RMSE
-11.038710923489962

0.8.3 Lasso Regression

[55]:
```
pipe_LAR = Pipeline([('model', Lasso(random_state=42))])
param_grid = {
    'model__alpha': [10.0**x for x in np.arange(-4,4)]
}
```

[56]:
```
print("Testing bike..\n")
gridBikeLAR_F = GridSearchCV(pipe_LAR, param_grid=param_grid, cv=10, n_jobs=-1,␣
 ↪verbose=1,
                     scoring='neg_root_mean_squared_error',␣
 ↪return_train_score=True).fit(XBikeCur_F, YBike)
gridBikeLAR_FS = GridSearchCV(pipe_LAR, param_grid=param_grid, cv=10,␣
 ↪n_jobs=-1, verbose=1,
                     scoring='neg_root_mean_squared_error',␣
 ↪return_train_score=True).fit(XBikeCur_FS, YBike)
gridBikeLAR_M = GridSearchCV(pipe_LAR, param_grid=param_grid, cv=10, n_jobs=-1,␣
 ↪verbose=1,
                     scoring='neg_root_mean_squared_error',␣
 ↪return_train_score=True).fit(XBikeCur_MIR, YBike)
gridBikeLAR_MS = GridSearchCV(pipe_LAR, param_grid=param_grid, cv=10,␣
 ↪n_jobs=-1, verbose=1,
                     scoring='neg_root_mean_squared_error',␣
 ↪return_train_score=True).fit(XBikeCur_MIRS, YBike)
print("Testing suicide..\n")
gridSuicideLAR_F = GridSearchCV(pipe_LAR, param_grid=param_grid, cv=10,␣
 ↪n_jobs=-1, verbose=1,
                     scoring='neg_root_mean_squared_error',␣
 ↪return_train_score=True).fit(XSuicideCur_F, YSuicide)
```

```
gridSuicideLAR_FS = GridSearchCV(pipe_LAR, param_grid=param_grid, cv=10,␣
↪n_jobs=-1, verbose=1,
                    scoring='neg_root_mean_squared_error',␣
↪return_train_score=True).fit(XSuicideCur_FS, YSuicide)
print("Testing transcoding..\n")
gridTranscodeLAR_F = GridSearchCV(pipe_LAR, param_grid=param_grid, cv=10,␣
↪n_jobs=-1, verbose=1,
                    scoring='neg_root_mean_squared_error',␣
↪return_train_score=True).fit(XTranscodeCur_F, YTranscode)
gridTranscodeLAR_FS = GridSearchCV(pipe_LAR, param_grid=param_grid, cv=10,␣
↪n_jobs=-1, verbose=1,
                    scoring='neg_root_mean_squared_error',␣
↪return_train_score=True).fit(XTranscodeCur_FS, YTranscode)
```

```
Testing bike..

Fitting 10 folds for each of 8 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    1.7s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    0.1s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 8 candidates, totalling 80 fits
Fitting 10 folds for each of 8 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    0.1s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    0.1s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 8 candidates, totalling 80 fits
Testing suicide..

Fitting 10 folds for each of 8 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    0.2s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 8 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    0.2s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Testing transcoding..

Fitting 10 folds for each of 8 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:   12.9s finished

Fitting 10 folds for each of 8 candidates, totalling 80 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    3.2s finished
```

[57]:
```
print('• No standardization, bike dataset, F1, lasso reg., Test RMSE:
 ↪',gridBikeLAR_F.best_score_,
      ',alpha:',gridBikeLAR_F.best_params_,'train RMSE',max(gridBikeLAR_F.
 ↪cv_results_['mean_train_score']))
print('• Standardization, bike dataset, F1, lasso reg., Test RMSE:
 ↪',gridBikeLAR_FS.best_score_,
      ',alpha:',gridBikeLAR_FS.best_params_,'train RMSE',max(gridBikeLAR_FS.
 ↪cv_results_['mean_train_score']))
print('• No standardization, bike dataset, MI, lasso reg., Test RMSE:
 ↪',gridBikeLAR_M.best_score_,
      ',alpha:',gridBikeLAR_F.best_params_,'train RMSE',max(gridBikeLAR_F.
 ↪cv_results_['mean_train_score']))
print('• Standardization, bike dataset, MI, lasso reg., Test RMSE:
 ↪',gridBikeLAR_MS.best_score_,
      ',alpha:',gridBikeLAR_MS.best_params_,'train RMSE',max(gridBikeLAR_MS.
 ↪cv_results_['mean_train_score']))
print('• No standardization, suicide dataset, F1, lasso reg., Test RMSE:
 ↪',gridSuicideLAR_F.best_score_,
      ',alpha:',gridSuicideLAR_F.best_params_,'train RMSE',max(gridSuicideLAR_F.
 ↪cv_results_['mean_train_score']))
print('• Standardization, suicide dataset, F1, lasso reg., Test RMSE:
 ↪',gridSuicideLAR_FS.best_score_,
      ',alpha:',gridSuicideLAR_FS.best_params_,'train␣
 ↪RMSE',max(gridSuicideLAR_FS.cv_results_['mean_train_score']))
print('• No standardization, transcoding dataset, F1, lasso reg., Test RMSE:
 ↪',gridTranscodeLAR_F.best_score_,
      ',alpha:',gridTranscodeLAR_F.best_params_,'train␣
 ↪RMSE',max(gridTranscodeLAR_F.cv_results_['mean_train_score']))
print('• Standardization, transcoding dataset, F1, lasso reg., Test RMSE:
 ↪',gridTranscodeLAR_FS.best_score_,
      ',alpha:',gridTranscodeLAR_FS.best_params_,'train␣
 ↪RMSE',max(gridTranscodeLAR_FS.cv_results_['mean_train_score']))
```

```
• No standardization, bike dataset, F1, lasso reg., Test RMSE:
-899.5074463695244 ,alpha: {'model__alpha': 10.0} train RMSE -836.3621040837518
• Standardization, bike dataset, F1, lasso reg., Test RMSE: -897.4951623793668
,alpha: {'model__alpha': 10.0} train RMSE -836.3621040459991
• No standardization, bike dataset, MI, lasso reg., Test RMSE:
-952.3265864115558 ,alpha: {'model__alpha': 10.0} train RMSE -836.3621040837518
• Standardization, bike dataset, MI, lasso reg., Test RMSE: -946.5350865872509
,alpha: {'model__alpha': 100.0} train RMSE -871.9979499730189
• No standardization, suicide dataset, F1, lasso reg., Test RMSE:
-15.330688655388517 ,alpha: {'model__alpha': 0.01} train RMSE
-15.420182168435108
```

• Standardization, suicide dataset, F1, lasso reg., Test RMSE:
-15.32750979337663 ,alpha: {'model__alpha': 0.1} train RMSE -15.42018215325216
• No standardization, transcoding dataset, F1, lasso reg., Test RMSE:
-11.059330236660127 ,alpha: {'model__alpha': 0.01} train RMSE -11.03871093075142
• Standardization, transcoding dataset, F1, lasso reg., Test RMSE:
-11.054351412935565 ,alpha: {'model__alpha': 0.1} train RMSE -11.038711188883545

0.8.4 p-value example

[58]:
```
p_ex = OLS(YSuicide, suicide_LR.loc[:, suicide_LR.columns != 'suicides/100k␣
 ↪pop']).fit()
print(p_ex.pvalues.sort_values(ascending=True))
```

```
gdp_per_capita ($)            8.766941e-48
 gdp_for_year ($)             1.003279e-11
age_75+ years                 1.078009e-06
Continent_EU                  4.850388e-06
age_55-74 years               7.534267e-05
sex_male                      9.415642e-05
generation_G.I. Generation    1.003124e-04
age_35-54 years               1.274671e-04
Continent_OC                  1.912970e-04
Continent_AS                  3.764538e-04
generation_Boomers            3.824482e-04
generation_Generation Z       4.133131e-04
generation_Silent             4.412699e-04
age_25-34 years               4.422605e-04
Continent_SA                  4.666982e-04
generation_Generation X       4.824022e-04
generation_Millenials         6.498364e-04
year                          1.026107e-03
sex_female                    1.279165e-03
age_15-24 years               1.978574e-03
Continent_AF                  2.456121e-03
Continent_NA                  2.921954e-03
population                    7.801250e-03
age_5-14 years                6.607614e-02
dtype: float64
```

0.9 Question 14 to 16

0.9.1 Finding optimal degree

[59]:
```
degree_list = np.arange(1,11,1)

pipe_PR_bike = Pipeline([
    ('PR', PolynomialFeatures()),
    ('model', Ridge(random_state=42))
```

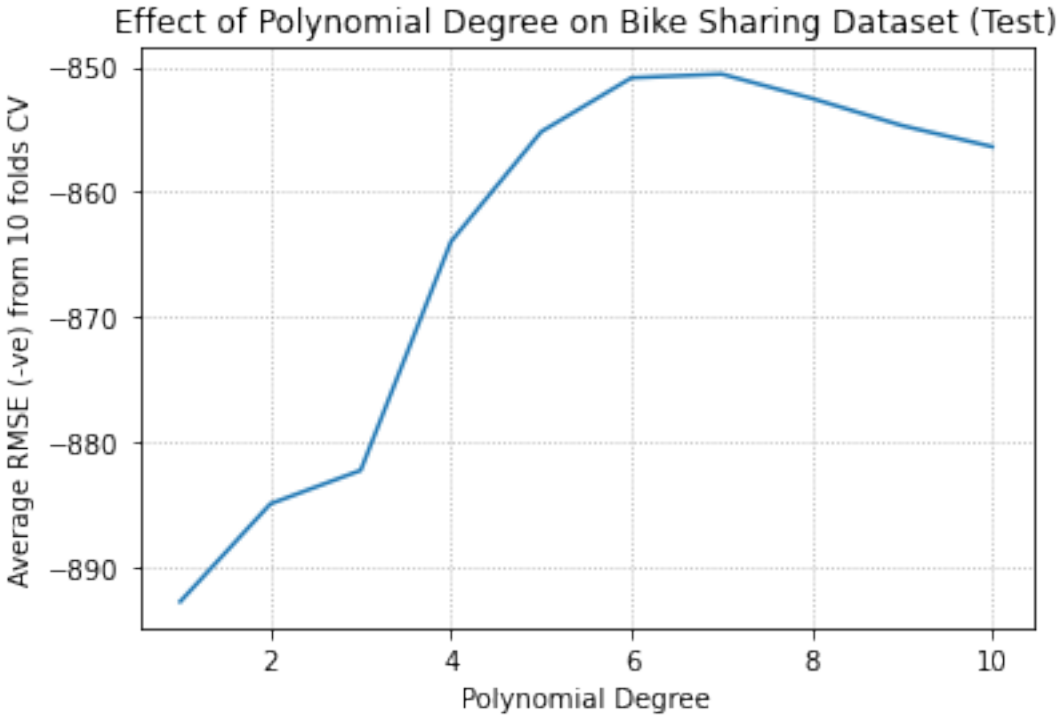
```
])

pipe_PR_suicide = Pipeline([
    ('PR', PolynomialFeatures()),
    ('model', Ridge(random_state=42))
])

pipe_PR_transcode = Pipeline([
    ('PR', PolynomialFeatures()),
    ('model', Ridge(random_state=42))
])

param_grid_PR = {
    'PR__degree': degree_list,
    'model__alpha': [10.0**x for x in np.arange(-4,4)]

}
```

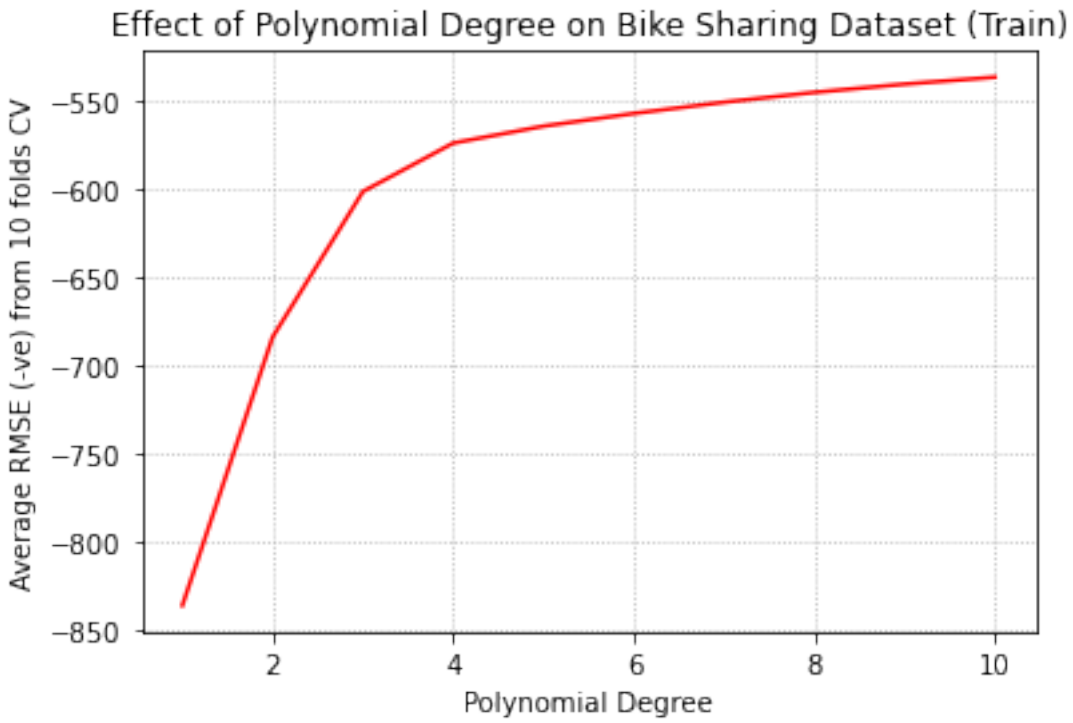
[60]:
```
gridbike_PR = GridSearchCV(pipe_PR_bike, param_grid=param_grid_PR, cv=10,␣
↪n_jobs=-1, verbose=1,
                    scoring='neg_root_mean_squared_error',␣
↪return_train_score=True).fit(XBikeCur_F,YBike)
```

```
Fitting 10 folds for each of 80 candidates, totalling 800 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done 144 tasks      | elapsed:    0.5s
[Parallel(n_jobs=-1)]: Done 800 out of 800 | elapsed:  2.1min finished
```

[69]:
```
poly_result = pd.DataFrame(gridbike_PR.
↪cv_results_)[['mean_test_score','mean_train_score','param_PR__degree','param_model__alpha']]
bike_score = []
bike_train = []
bike_alpha = []
for i in degree_list:
    bike_score.append((poly_result.loc[poly_result['param_PR__degree'] == i]).
↪max().mean_test_score)
    bike_train.append((poly_result.loc[poly_result['param_PR__degree'] == i]).
↪max().mean_train_score)
    bike_alpha.append(float(poly_result['param_model__alpha'][
        (poly_result.loc[poly_result['param_PR__degree'] == i])
        [['mean_test_score']].idxmax()].to_numpy()))
plt.plot(degree_list,bike_score)
plt.grid(linestyle=':')
plt.xlabel('Polynomial Degree')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of Polynomial Degree on Bike Sharing Dataset (Test)')
```

```
plt.savefig('Q15a.png',dpi=300,bbox_inches='tight')
plt.show()
plt.plot(degree_list,bike_train,'r')
plt.grid(linestyle=':')
plt.xlabel('Polynomial Degree')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of Polynomial Degree on Bike Sharing Dataset (Train)')
plt.savefig('Q15b.png',dpi=300,bbox_inches='tight')
plt.show()
```



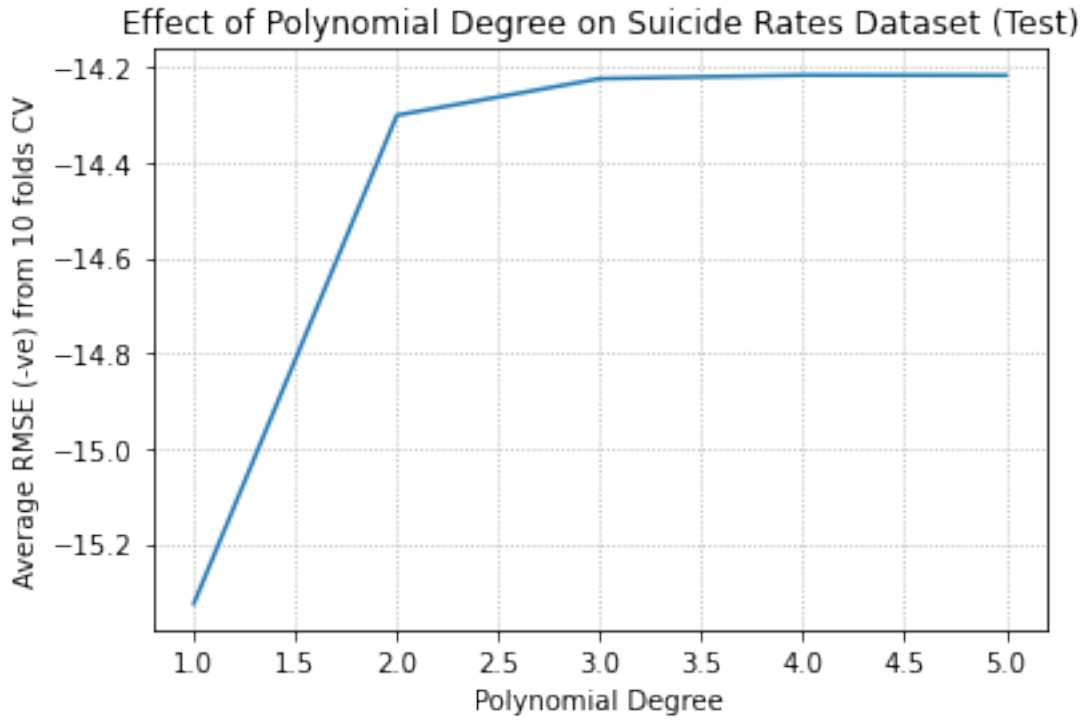


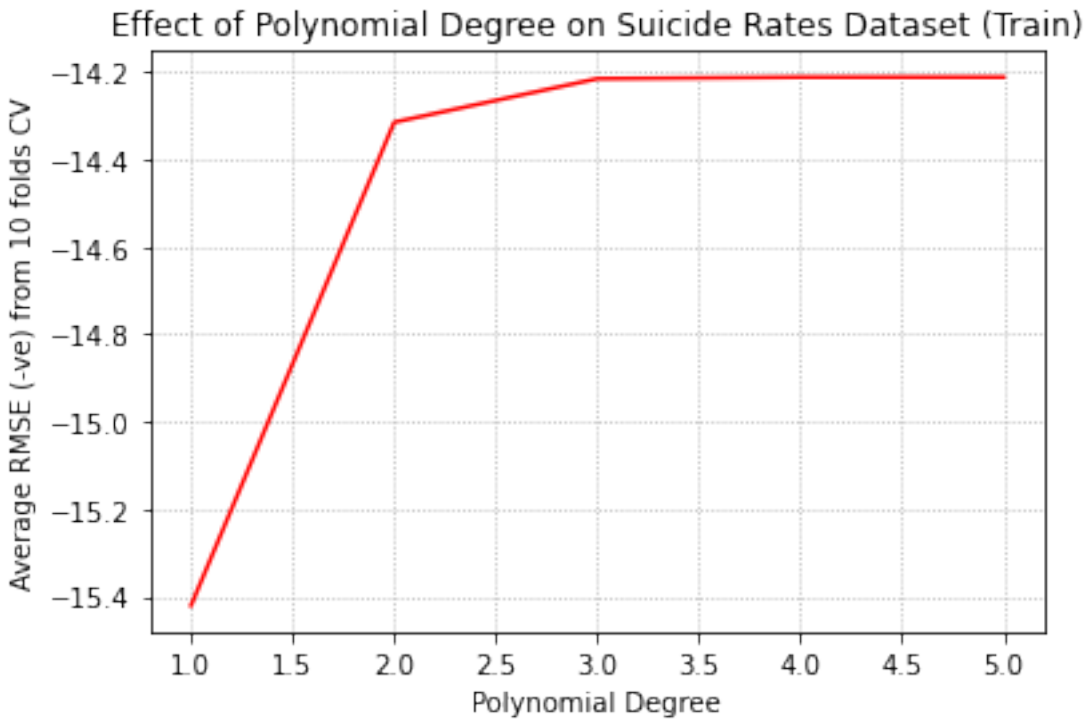
[70]:
```
degree_list = np.arange(1,6,1)
param_grid_PR = {
    'PR__degree': degree_list,
    'model__alpha': [10.0**x for x in np.arange(-4,4)]

}

gridSuicide_PR = GridSearchCV(pipe_PR_suicide, param_grid=param_grid_PR, cv=10,␣
 ↪n_jobs=-1, verbose=1,
                      scoring='neg_root_mean_squared_error',␣
 ↪return_train_score=True).fit(XSuicideCur_F,YSuicide)
```

```
Fitting 10 folds for each of 40 candidates, totalling 400 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done 224 tasks      | elapsed:    7.3s
[Parallel(n_jobs=-1)]: Done 400 out of 400 | elapsed:  1.6min finished
```

[71]:
```
poly_result = pd.DataFrame(gridSuicide_PR.
 ↪cv_results_)[['mean_test_score','mean_train_score','param_PR__degree','param_model__alpha']]
suicide_score = []
suicide_train = []
suicide_alpha = []
for i in degree_list:
```

```
    suicide_score.append((poly_result.loc[poly_result['param_PR__degree'] ==␣
↪i]).max().mean_test_score)
    suicide_train.append((poly_result.loc[poly_result['param_PR__degree'] ==␣
↪i]).max().mean_train_score)
    suicide_alpha.append(float(poly_result['param_model__alpha'][
        (poly_result.loc[poly_result['param_PR__degree'] == i])
        [['mean_test_score']].idxmax()].to_numpy()))
plt.plot(degree_list,suicide_score)
plt.grid(linestyle=':')
plt.xlabel('Polynomial Degree')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of Polynomial Degree on Suicide Rates Dataset (Test)')
plt.savefig('Q15c.png',dpi=300,bbox_inches='tight')
plt.show()
plt.plot(degree_list,suicide_train,'r')
plt.grid(linestyle=':')
plt.xlabel('Polynomial Degree')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of Polynomial Degree on Suicide Rates Dataset (Train)')
plt.savefig('Q15d.png',dpi=300,bbox_inches='tight')
plt.show()
```

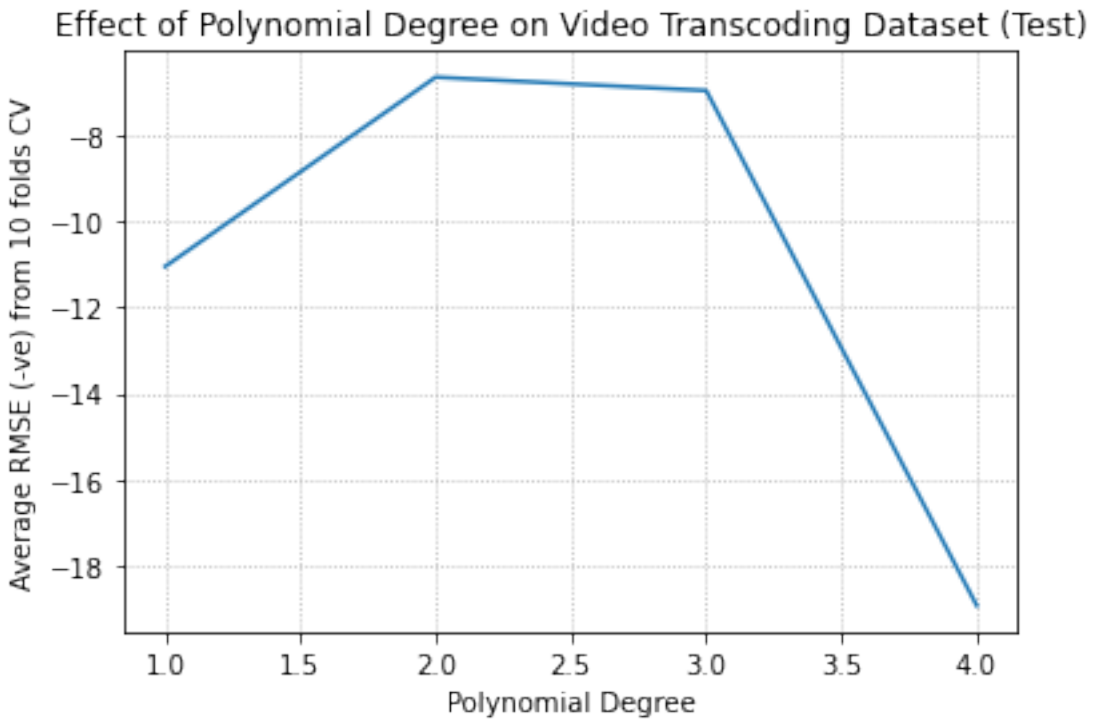


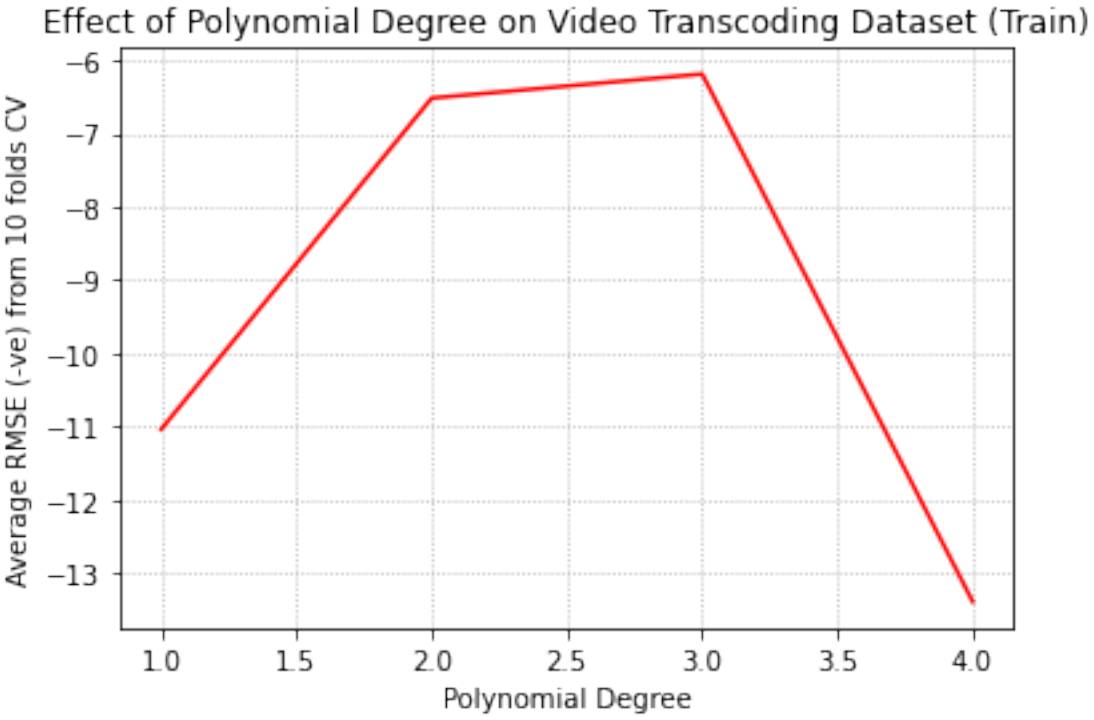


[72]:
```
degree_list = np.arange(1,5,1)
param_grid_PR = {
    'PR__degree': degree_list,
    'model__alpha': [10.0**x for x in np.arange(-4,4)]

}

gridTranscode_PR = GridSearchCV(pipe_PR_transcode, param_grid=param_grid_PR,␣
↪cv=10, n_jobs=-1, verbose=1,
                    scoring='neg_root_mean_squared_error',␣
↪return_train_score=True).fit(XTranscodeCur_F,YTranscode)
```

```
Fitting 10 folds for each of 32 candidates, totalling 320 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done 208 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 257 out of 320 | elapsed:  3.7min remaining:   54.8s
[Parallel(n_jobs=-1)]: Done 320 out of 320 | elapsed:  6.8min finished
/home/nesl/anaconda3/envs/tflite/lib/python3.8/site-
packages/sklearn/linear_model/_ridge.py:147: LinAlgWarning: Ill-conditioned
matrix (rcond=1.6857e-30): result may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
```

[73]:
```
poly_result = pd.DataFrame(gridTranscode_PR.
 ↪cv_results_)[['mean_test_score','mean_train_score','param_PR__degree','param_model__alpha']]
transcode_score = []
transcode_train = []
transcode_alpha = []
for i in degree_list:
    transcode_score.append((poly_result.loc[poly_result['param_PR__degree'] ==␣
 ↪i]).max().mean_test_score)
    transcode_train.append((poly_result.loc[poly_result['param_PR__degree'] ==␣
 ↪i]).max().mean_train_score)
    transcode_alpha.append(float(poly_result['param_model__alpha'][
        (poly_result.loc[poly_result['param_PR__degree'] == i])
        [['mean_test_score']].idxmax()].to_numpy()))
plt.plot(degree_list,transcode_score)
plt.grid(linestyle=':')
plt.xlabel('Polynomial Degree')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of Polynomial Degree on Video Transcoding Dataset (Test)')
plt.savefig('Q15e.png',dpi=300,bbox_inches='tight')
plt.show()
plt.plot(degree_list,transcode_train,'r')
plt.grid(linestyle=':')
plt.xlabel('Polynomial Degree')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of Polynomial Degree on Video Transcoding Dataset (Train)')
plt.savefig('Q15f.png',dpi=300,bbox_inches='tight')
plt.show()
```


Effect of Polynomial Degree on Video Transcoding Dataset (Test)
Average RMSE (-ve) from 10 folds CV
−8
−10
−12
−14
−16
−18
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Polynomial Degree


Effect of Polynomial Degree on Video Transcoding Dataset (Train)
Average RMSE (-ve) from 10 folds CV
−6
−7
−8
−9
−10
−11
−12
−13
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Polynomial Degree

0.9.2 Most Salient Features

[74]:
```
chY = SelectKBest(score_func=f_regression, k=10)
XTranscode_Test = chY.fit_transform(bike_LR.loc[:, bike_LR.columns != 'cnt'],␣
 ↪bike_LR.cnt)
column_names = bike_LR.loc[:, bike_LR.columns != 'cnt'].columns[chY.
 ↪get_support()]

b_params = gridbike_PR.best_estimator_.get_params()
b_coefs = b_params['model'].coef_
b_feature_name = list(column_names)
b_names = b_params['PR'].get_feature_names(b_feature_name)
b_sorted_indice = np.argsort(-abs(b_coefs))
salient_features =[b_names[i] for i in b_sorted_indice[:5]]
print ('Top 5 Salient features (bike):',salient_features)
```

```
Top 5 Salient features (bike): ['temp', 'atemp', 'windspeed', 'atemp^2', 'temp
atemp']
```

[75]:
```
chY = SelectKBest(score_func=f_regression, k=10)
XTranscode_Test = chY.fit_transform(suicide_LR.loc[:, suicide_LR.columns !=␣
 ↪'suicides/100k pop'], suicide_LR["suicides/100k pop"])
column_names = suicide_LR.loc[:, suicide_LR.columns != 'suicides/100k pop'].
 ↪columns[chY.get_support()]

s_params = gridSuicide_PR.best_estimator_.get_params()
s_coefs = s_params['model'].coef_
s_feature_name = list(column_names)
s_names = s_params['PR'].get_feature_names(s_feature_name)
s_sorted_indice = np.argsort(-abs(s_coefs))
salient_feature =[s_names[i] for i in s_sorted_indice[:5]]
print ('Top 5 Salient features (suicide):',salient_feature)
```

```
Top 5 Salient features (suicide): ['sex_female', 'sex_male', 'sex_female^2',
'sex_female^3', 'sex_male^2']
```

[76]:
```
chY = SelectKBest(score_func=f_regression, k=10)
XTranscode_Test = chY.fit_transform(transcode_LR.loc[:,  transcode_LR.columns !
 ↪= 'utime'], transcode_LR["utime"])
column_names = transcode_LR.loc[:,  transcode_LR.columns != 'utime'].
 ↪columns[chY.get_support()]

t_params = gridTranscode_PR.best_estimator_.get_params()
t_coefs = t_params['model'].coef_
t_feature_name = list(column_names)
t_names = t_params['PR'].get_feature_names(t_feature_name)
t_sorted_indice = np.argsort(-abs(t_coefs))
```

```
salient_feature =[t_names[i] for i in t_sorted_indice[:5]]
print ('Top 5 Salient features (transcoding):',salient_feature)
```

```
Top 5 Salient features (transcoding): ['o_codec_flv', 'o_codec_flv^2',
'o_codec_h264^2', 'o_codec_h264', 'o_codec_mpeg4']
```

0.9.3 Testing Inverse Features

[77]:
```
chY = SelectKBest(score_func=f_regression, k=10)
XTranscode_Test = chY.fit_transform(transcode_LR.loc[:,  transcode_LR.columns !
 ↪= 'utime'], transcode_LR["utime"])
column_names = transcode_LR.loc[:,  transcode_LR.columns != 'utime'].
 ↪columns[chY.get_support()]
XTranscode_Test = pd.DataFrame(XTranscode_Test,columns=list(column_names))
print(list(column_names))
```

```
['width', 'height', 'bitrate', 'o_bitrate', 'o_framerate', 'o_width',
'o_height', 'o_codec_flv', 'o_codec_h264', 'o_codec_mpeg4']
```

[78]:
```
inv_feat = np.divide(np.
 ↪prod(XTranscode_Test[['o_width','o_height']],axis=1),XTranscode_Test['o_bitrate'])
XTranscode_Test['inv_feat'] = inv_feat
print(XTranscode_Test)
```

```
        width  height    bitrate  o_bitrate  o_framerate  o_width  o_height  \
0       176.0   144.0    54590.0    56000.0        12.00    176.0     144.0
1       176.0   144.0    54590.0    56000.0        12.00    320.0     240.0
2       176.0   144.0    54590.0    56000.0        12.00    480.0     360.0
3       176.0   144.0    54590.0    56000.0        12.00    640.0     480.0
4       176.0   144.0    54590.0    56000.0        12.00   1280.0     720.0
…         …     …          …          …             …        …         …
68779   480.0   360.0   278822.0   242000.0        24.00    640.0     480.0
68780   640.0   480.0   639331.0   539000.0        29.97   1920.0    1080.0
68781   320.0   240.0   359345.0   539000.0        12.00    176.0     144.0
68782  1280.0   720.0  2847539.0   539000.0        12.00    320.0     240.0
68783   176.0   144.0    55242.0   820000.0        24.00    176.0     144.0

       o_codec_flv  o_codec_h264  o_codec_mpeg4   inv_feat
0              0.0           0.0            1.0   0.452571
1              0.0           0.0            1.0   1.371429
2              0.0           0.0            1.0   3.085714
3              0.0           0.0            1.0   5.485714
4              0.0           0.0            1.0  16.457143
…            …             …              …        …
68779          1.0           0.0            0.0   1.269421
68780          0.0           0.0            1.0   3.847124
68781          1.0           0.0            0.0   0.047020
```

```
68782          0.0           0.0            1.0   0.142486
68783          0.0           1.0            0.0   0.030907

[68784 rows x 11 columns]
```

[79]:
```
degree_list = [2]
param_grid_PR = {
    'PR__degree': degree_list,
    'model__alpha': [transcode_alpha[1]]

}

gridTranscode_PR_test = GridSearchCV(pipe_PR_transcode,␣
 ↪param_grid=param_grid_PR, cv=10, n_jobs=-1, verbose=1,
                      scoring='neg_root_mean_squared_error',␣
 ↪return_train_score=True).fit(XTranscode_Test ,YTranscode)
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    0.3s remaining:    1.2s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    0.4s finished
/home/nesl/anaconda3/envs/tflite/lib/python3.8/site-
packages/sklearn/linear_model/_ridge.py:147: LinAlgWarning: Ill-conditioned
matrix (rcond=1.6857e-30): result may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
```

[80]:
```
print('Average Test RMSE (-ve) without inverse feature (degree = 2):
 ↪',transcode_score[1])
print('Average Test RMSE (-ve) with inverse feature (degree = 2):
 ↪',gridTranscode_PR_test.best_score_)
```

```
Average Test RMSE (-ve) without inverse feature (degree = 2): -6.669831310279045
Average Test RMSE (-ve) with inverse feature (degree = 2): -6.063805812065415
```

0.10 Question 17 to 20

[163]:
```
a_list = [10,20,30,50]
all_combinations = []
for r in range(len(a_list) + 1):
    combinations_object = itertools.combinations_with_replacement(a_list, r)
    combinations_list = list(combinations_object)
    all_combinations += combinations_list
all_combinations = all_combinations[1:]

pipe_NN = Pipeline([
    ('model', MLPRegressor(random_state=42,max_iter=1000))
])
```

```
param_grid_NN = {
    'model__hidden_layer_sizes': all_combinations,
    'model__alpha': [10.0**x for x in np.arange(-4,2)],
    'model__activation': ['logistic', 'tanh', 'relu']
}
```

[164]:
```
print(all_combinations)
```

```
[(10,), (20,), (30,), (50,), (10, 10), (10, 20), (10, 30), (10, 50), (20, 20),
(20, 30), (20, 50), (30, 30), (30, 50), (50, 50), (10, 10, 10), (10, 10, 20),
(10, 10, 30), (10, 10, 50), (10, 20, 20), (10, 20, 30), (10, 20, 50), (10, 30,
30), (10, 30, 50), (10, 50, 50), (20, 20, 20), (20, 20, 30), (20, 20, 50), (20,
30, 30), (20, 30, 50), (20, 50, 50), (30, 30, 30), (30, 30, 50), (30, 50, 50),
(50, 50, 50), (10, 10, 10, 10), (10, 10, 10, 20), (10, 10, 10, 30), (10, 10, 10,
50), (10, 10, 20, 20), (10, 10, 20, 30), (10, 10, 20, 50), (10, 10, 30, 30),
(10, 10, 30, 50), (10, 10, 50, 50), (10, 20, 20, 20), (10, 20, 20, 30), (10, 20,
20, 50), (10, 20, 30, 30), (10, 20, 30, 50), (10, 20, 50, 50), (10, 30, 30, 30),
(10, 30, 30, 50), (10, 30, 50, 50), (10, 50, 50, 50), (20, 20, 20, 20), (20, 20,
20, 30), (20, 20, 20, 50), (20, 20, 30, 30), (20, 20, 30, 50), (20, 20, 50, 50),
(20, 30, 30, 30), (20, 30, 30, 50), (20, 30, 50, 50), (20, 50, 50, 50), (30, 30,
30, 30), (30, 30, 30, 50), (30, 30, 50, 50), (30, 50, 50, 50), (50, 50, 50, 50)]
```

[165]:
```
gridbike_NN = GridSearchCV(pipe_NN, param_grid=param_grid_NN, cv=10, n_jobs=-1,␣
↪verbose=1,
                                   scoring='neg_root_mean_squared_error',␣
↪return_train_score=True).fit(XBikeCur_F, YBike)
```

```
Fitting 10 folds for each of 1242 candidates, totalling 12420 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done 136 tasks      | elapsed:   20.7s
[Parallel(n_jobs=-1)]: Done 386 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 736 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done 1186 tasks      | elapsed:  4.0min
[Parallel(n_jobs=-1)]: Done 1736 tasks      | elapsed:  5.8min
[Parallel(n_jobs=-1)]: Done 2386 tasks      | elapsed:  8.1min
[Parallel(n_jobs=-1)]: Done 3136 tasks      | elapsed: 10.6min
[Parallel(n_jobs=-1)]: Done 3986 tasks      | elapsed: 13.6min
[Parallel(n_jobs=-1)]: Done 4936 tasks      | elapsed: 17.2min
[Parallel(n_jobs=-1)]: Done 5986 tasks      | elapsed: 21.0min
[Parallel(n_jobs=-1)]: Done 7136 tasks      | elapsed: 25.4min
[Parallel(n_jobs=-1)]: Done 8386 tasks      | elapsed: 30.1min
[Parallel(n_jobs=-1)]: Done 9736 tasks      | elapsed: 33.6min
[Parallel(n_jobs=-1)]: Done 11186 tasks      | elapsed: 37.3min
[Parallel(n_jobs=-1)]: Done 12420 out of 12420 | elapsed: 40.5min finished
/home/nesl/anaconda3/envs/tflite/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:582:
```

```
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[182]:
```
poly_result = pd.DataFrame(gridbike_NN.
 ↪cv_results_)[['mean_test_score','mean_train_score','param_model__alpha','param_model__activ
print('Best parameters (bike):',gridbike_NN.best_params_,',Test RMSE:
 ↪',gridbike_NN.best_score_)
print('Train RMSE:',max(poly_result.mean_train_score))
```

```
Best parameters (bike): {'model__activation': 'relu', 'model__alpha': 10.0,
'model__hidden_layer_sizes': (10, 30, 30, 30)} ,Test RMSE: -859.001551074912
Train RMSE: -763.5589843269157
```

[183]:
```
gridSuicide_NN = GridSearchCV(pipe_NN, param_grid=param_grid_NN, cv=10,␣
 ↪n_jobs=-1, verbose=1,
                                  scoring='neg_root_mean_squared_error',␣
 ↪return_train_score=True).fit(XSuicideCur_F, YSuicide)
```

```
Fitting 10 folds for each of 1242 candidates, totalling 12420 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done 136 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done 386 tasks      | elapsed:  5.2min
[Parallel(n_jobs=-1)]: Done 736 tasks      | elapsed: 10.7min
[Parallel(n_jobs=-1)]: Done 1186 tasks      | elapsed: 16.5min
[Parallel(n_jobs=-1)]: Done 1736 tasks      | elapsed: 24.2min
[Parallel(n_jobs=-1)]: Done 2386 tasks      | elapsed: 33.2min
[Parallel(n_jobs=-1)]: Done 3136 tasks      | elapsed: 44.1min
[Parallel(n_jobs=-1)]: Done 3986 tasks      | elapsed: 56.6min
[Parallel(n_jobs=-1)]: Done 4936 tasks      | elapsed: 67.6min
[Parallel(n_jobs=-1)]: Done 5986 tasks      | elapsed: 80.0min
[Parallel(n_jobs=-1)]: Done 7136 tasks      | elapsed: 94.1min
[Parallel(n_jobs=-1)]: Done 8386 tasks      | elapsed: 110.9min
[Parallel(n_jobs=-1)]: Done 9736 tasks      | elapsed: 118.6min
[Parallel(n_jobs=-1)]: Done 11186 tasks      | elapsed: 126.9min
[Parallel(n_jobs=-1)]: Done 12420 out of 12420 | elapsed: 136.6min finished
```

[184]:
```
poly_result = pd.DataFrame(gridSuicide_NN.
 ↪cv_results_)[['mean_test_score','mean_train_score','param_model__alpha','param_model__activ
print('Best parameters (suicide):',gridSuicide_NN.best_params_,',Test RMSE:
 ↪',gridSuicide_NN.best_score_)
print('Train RMSE:',max(poly_result.mean_train_score))
```

```
Best parameters (suicide): {'model__activation': 'relu', 'model__alpha': 0.001,
'model__hidden_layer_sizes': (20, 30, 50, 50)} ,Test RMSE: -14.107559037480362
Train RMSE: -14.216131503452496
```

[185]:
```
gridTranscode_NN = GridSearchCV(pipe_NN, param_grid=param_grid_NN, cv=10,␣
↪n_jobs=-1, verbose=1,
                                scoring='neg_root_mean_squared_error',␣
↪return_train_score=True).fit(XTranscodeCur_F, YTranscode)
```

```
Fitting 10 folds for each of 1242 candidates, totalling 12420 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done 136 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 386 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done 736 tasks      | elapsed:  9.1min
[Parallel(n_jobs=-1)]: Done 1186 tasks      | elapsed: 13.3min
[Parallel(n_jobs=-1)]: Done 1736 tasks      | elapsed: 19.5min
[Parallel(n_jobs=-1)]: Done 2386 tasks      | elapsed: 26.7min
[Parallel(n_jobs=-1)]: Done 3136 tasks      | elapsed: 35.6min
[Parallel(n_jobs=-1)]: Done 3986 tasks      | elapsed: 48.7min
[Parallel(n_jobs=-1)]: Done 4936 tasks      | elapsed: 60.5min
[Parallel(n_jobs=-1)]: Done 5986 tasks      | elapsed: 71.3min
[Parallel(n_jobs=-1)]: Done 7136 tasks      | elapsed: 83.3min
[Parallel(n_jobs=-1)]: Done 8386 tasks      | elapsed: 103.4min
[Parallel(n_jobs=-1)]: Done 9736 tasks      | elapsed: 115.7min
[Parallel(n_jobs=-1)]: Done 11186 tasks      | elapsed: 128.7min
[Parallel(n_jobs=-1)]: Done 12420 out of 12420 | elapsed: 140.2min finished
```

[189]:
```
poly_result = pd.DataFrame(gridTranscode_NN.
↪cv_results_)[['mean_test_score','mean_train_score','param_model__alpha','param_model__activ
print('Best parameters (suicide):',gridTranscode_NN.best_params_,',Test RMSE:
↪',gridTranscode_NN.best_score_)
print('Train RMSE:',max(poly_result.mean_train_score))
```

```
Best parameters (suicide): {'model__activation': 'tanh', 'model__alpha': 10.0,
'model__hidden_layer_sizes': (50, 50, 50, 50)} ,Test RMSE: -16.024952369303044
Train RMSE: -16.060624125626163
```

0.11 Question 21 to 23, 28

0.11.1 Optimal and effect of each hyperparameter

[249]:
```
pipe_RF = Pipeline([
    ('model', RandomForestRegressor(random_state=42, oob_score=True))
])

param_grid_RF = {
    'model__max_features': np.arange(1,11,1),
    'model__n_estimators': np.arange(10, 210, 10),
    'model__max_depth': np.arange(1, 20, 1)

}
```

[250]:
```
gridbike_RF = GridSearchCV(pipe_RF, param_grid=param_grid_RF, cv=10, n_jobs=-1,␣
↪verbose=1,
                                   scoring='neg_root_mean_squared_error',␣
↪return_train_score=True).fit(XBikeCur_F, YBike)
```

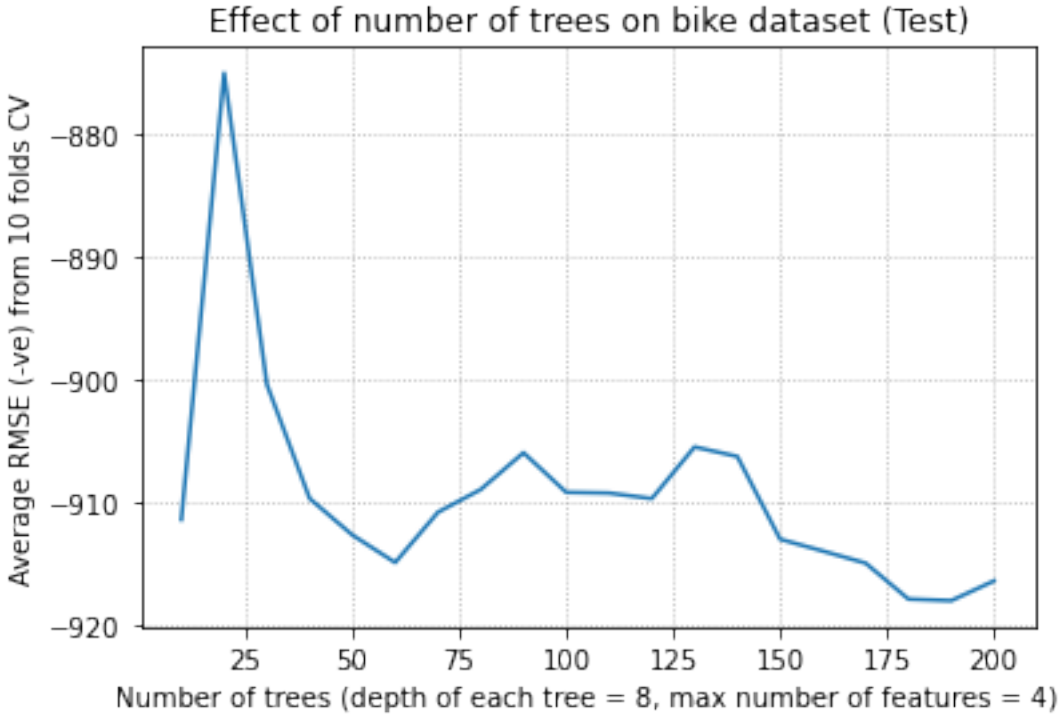
```
Fitting 10 folds for each of 3800 candidates, totalling 38000 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done 136 tasks      | elapsed:    2.7s
[Parallel(n_jobs=-1)]: Done 386 tasks      | elapsed:    4.6s
[Parallel(n_jobs=-1)]: Done 736 tasks      | elapsed:    7.0s
[Parallel(n_jobs=-1)]: Done 1186 tasks      | elapsed:   10.5s
[Parallel(n_jobs=-1)]: Done 1736 tasks      | elapsed:   14.5s
[Parallel(n_jobs=-1)]: Done 2386 tasks      | elapsed:   19.5s
[Parallel(n_jobs=-1)]: Done 3136 tasks      | elapsed:   25.1s
[Parallel(n_jobs=-1)]: Done 3986 tasks      | elapsed:   32.1s
[Parallel(n_jobs=-1)]: Done 4936 tasks      | elapsed:   39.2s
[Parallel(n_jobs=-1)]: Done 5986 tasks      | elapsed:   48.2s
[Parallel(n_jobs=-1)]: Done 7136 tasks      | elapsed:   57.3s
[Parallel(n_jobs=-1)]: Done 8386 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 9736 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 11186 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 12736 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done 14386 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done 16136 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done 17986 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done 19936 tasks      | elapsed:  3.0min
[Parallel(n_jobs=-1)]: Done 21986 tasks      | elapsed:  3.3min
[Parallel(n_jobs=-1)]: Done 24136 tasks      | elapsed:  3.7min
[Parallel(n_jobs=-1)]: Done 26386 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done 28736 tasks      | elapsed:  4.5min
[Parallel(n_jobs=-1)]: Done 31186 tasks      | elapsed:  5.0min
[Parallel(n_jobs=-1)]: Done 33736 tasks      | elapsed:  5.5min
[Parallel(n_jobs=-1)]: Done 36386 tasks      | elapsed:  6.0min
[Parallel(n_jobs=-1)]: Done 38000 out of 38000 | elapsed:  6.3min finished
```

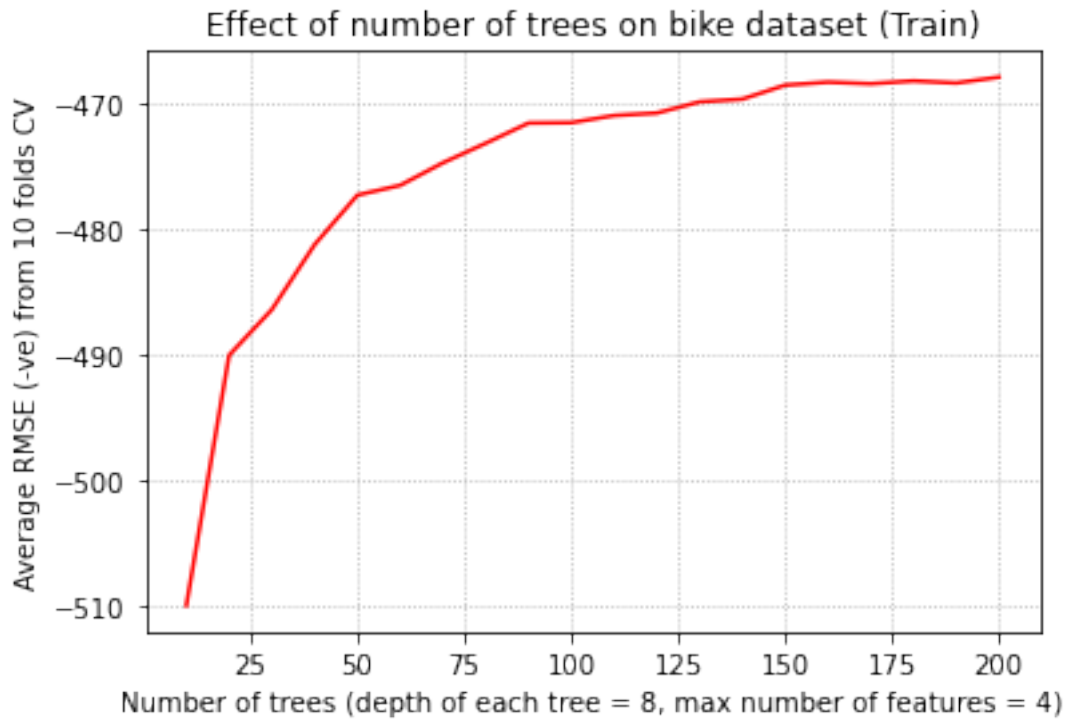
[251]:
```
poly_result = pd.DataFrame(gridbike_RF.
↪cv_results_)[['mean_test_score','mean_train_score','param_model__max_features','param_model
print('Best parameters (bike):',gridbike_RF.best_params_,',Test RMSE:
↪',gridbike_RF.best_score_)
print('Train RMSE:',max(poly_result.mean_train_score))
```

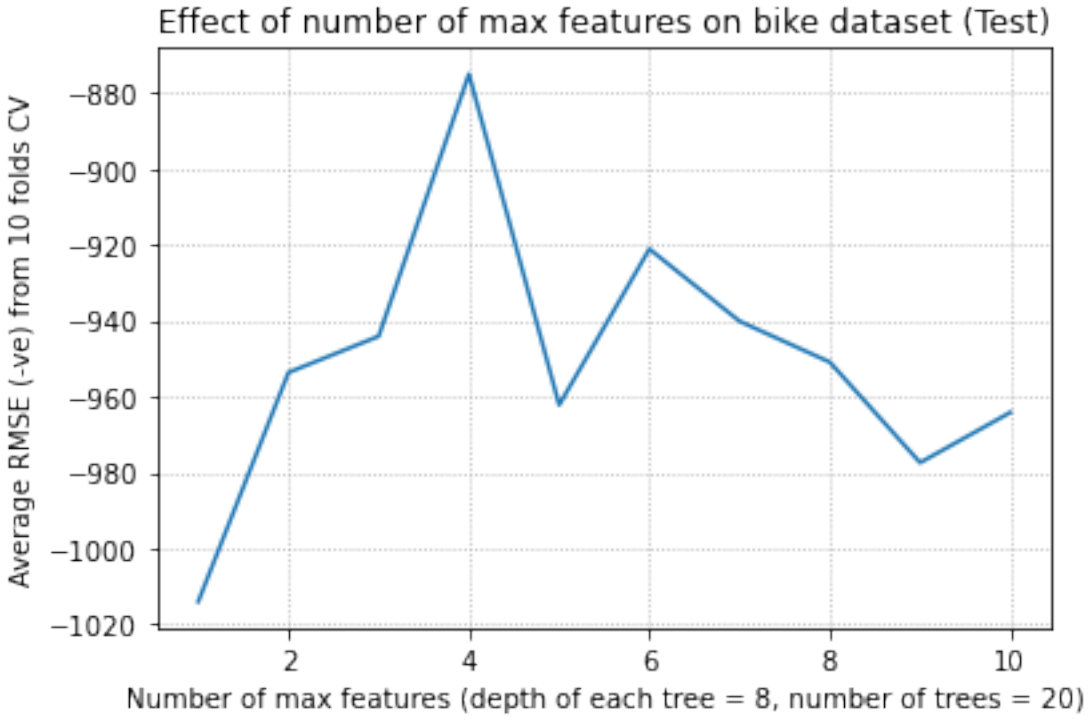
```
Best parameters (bike): {'model__max_depth': 8, 'model__max_features': 4,
'model__n_estimators': 20} ,Test RMSE: -875.1063994497797
Train RMSE: -276.8705434691136
```

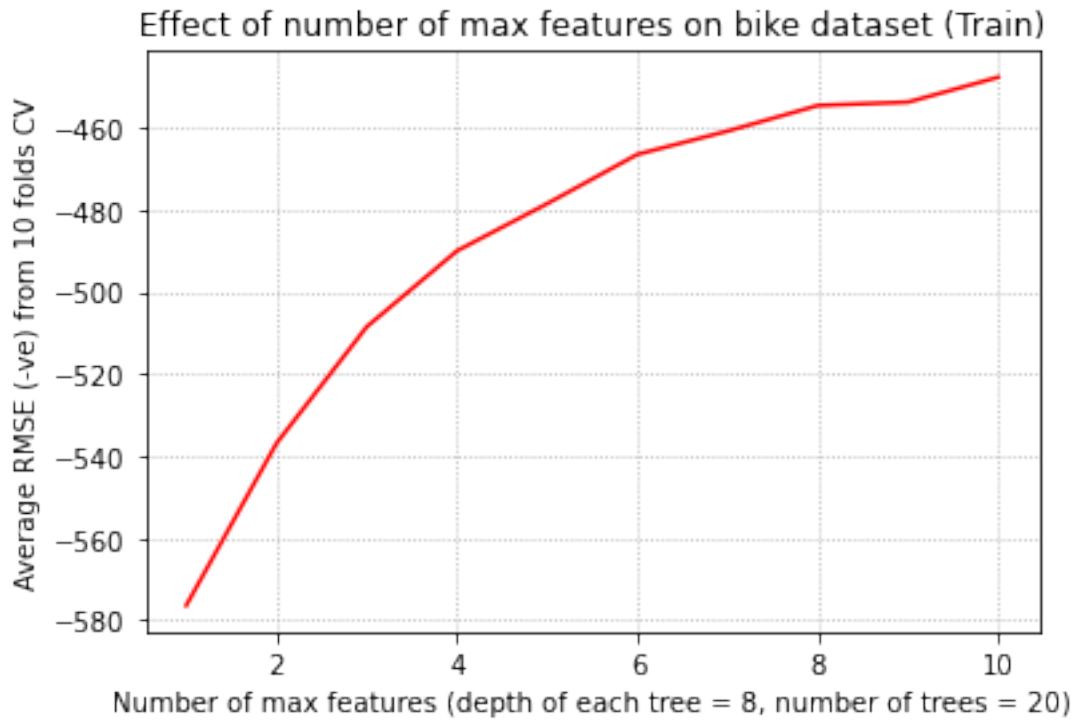
[252]:
```
max_features = np.arange(1,11,1).reshape(10)
n_estimators = np.arange(10, 210, 10).reshape(20)
max_depth = np.arange(1, 20, 1).reshape(19)

bike_score = list((poly_result[(poly_result['param_model__max_depth'] == 8) &␣
 ↪(poly_result['param_model__max_features'] == 4)]).mean_test_score)
bike_train = list((poly_result[(poly_result['param_model__max_depth'] == 8) &␣
 ↪(poly_result['param_model__max_features'] == 4)]).mean_train_score)
plt.plot(n_estimators,bike_score)
plt.grid(linestyle=':')
plt.xlabel('Number of trees (depth of each tree = 8, max number of features =␣
 ↪4)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of number of trees on bike dataset (Test)')
plt.savefig('Q21a.png',dpi=300,bbox_inches='tight')
plt.show()
plt.plot(n_estimators,bike_train,'r')
plt.grid(linestyle=':')
plt.xlabel('Number of trees (depth of each tree = 8, max number of features =␣
 ↪4)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of number of trees on bike dataset (Train)')
plt.savefig('Q21b.png',dpi=300,bbox_inches='tight')
plt.show()

bike_score = list((poly_result[(poly_result['param_model__max_depth'] == 8) &␣
 ↪(poly_result['param_model__n_estimators'] == 20)]).mean_test_score)
bike_train = list((poly_result[(poly_result['param_model__max_depth'] == 8) &␣
 ↪(poly_result['param_model__n_estimators'] == 20)]).mean_train_score)
plt.plot(max_features,bike_score)
plt.grid(linestyle=':')
plt.xlabel('Number of max features (depth of each tree = 8, number of trees =␣
 ↪20)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of number of max features on bike dataset (Test)')
plt.savefig('Q21c.png',dpi=300,bbox_inches='tight')
plt.show()
plt.plot(max_features,bike_train,'r')
plt.grid(linestyle=':')
plt.xlabel('Number of max features (depth of each tree = 8, number of trees =␣
 ↪20)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of number of max features on bike dataset (Train)')
plt.savefig('Q21d.png',dpi=300,bbox_inches='tight')
plt.show()
```

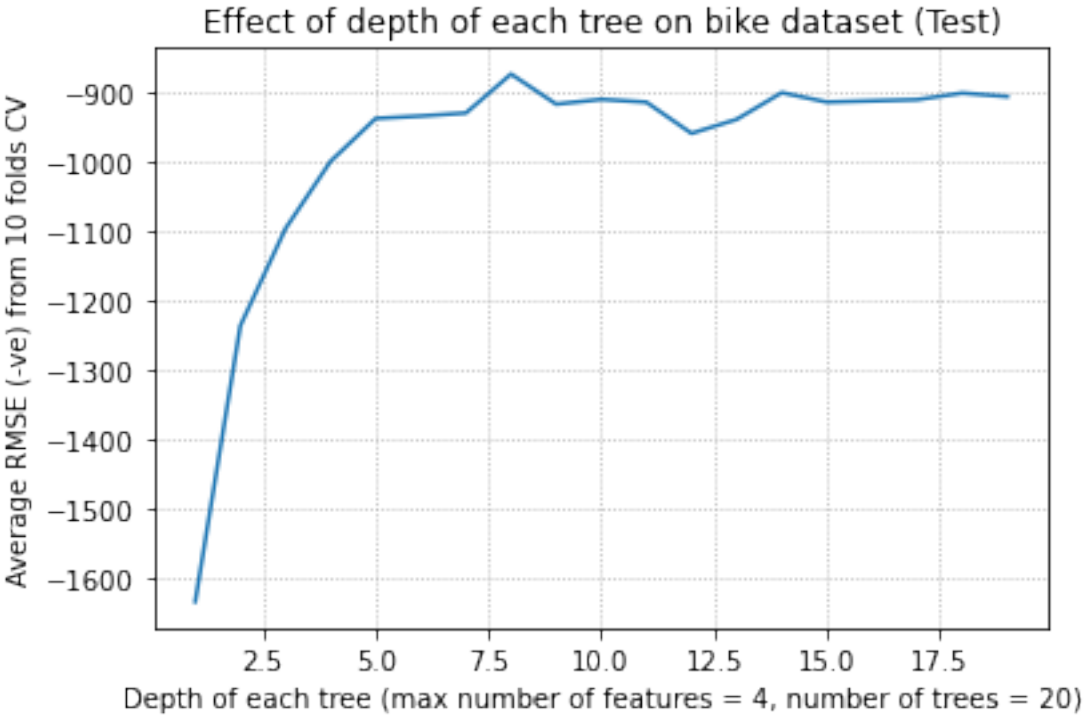
```
bike_score = list((poly_result[(poly_result['param_model__max_features'] == 4)␣
 ↪& (poly_result['param_model__n_estimators'] == 20)]).mean_test_score)
bike_train = list((poly_result[(poly_result['param_model__max_features'] == 4)␣
 ↪& (poly_result['param_model__n_estimators'] == 20)]).mean_train_score)
plt.plot(max_depth,bike_score)
plt.grid(linestyle=':')
plt.xlabel('Depth of each tree (max number of features = 4, number of trees =␣
 ↪20)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of depth of each tree on bike dataset (Test)')
plt.savefig('Q21e.png',dpi=300,bbox_inches='tight')
plt.show()
plt.plot(max_depth,bike_train,'r')
plt.grid(linestyle=':')
plt.xlabel('Depth of each tree  (max number of features = 4, number of trees =␣
 ↪20)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of depth of each tree on bike dataset (Train)')
plt.savefig('Q21f.png',dpi=300,bbox_inches='tight')
plt.show()
```

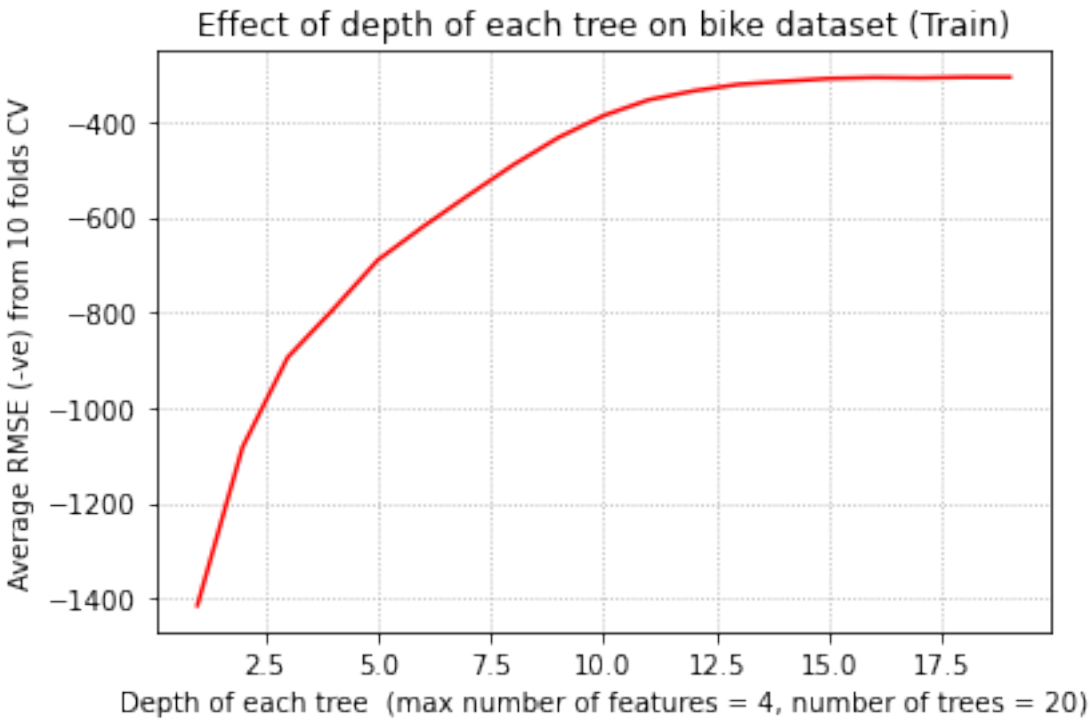



Effect of number of trees on bike dataset (Train)
Average RMSE (-ve) from 10 folds CV
−470
−480
−490
−500
−510
25
50
75
100
125
150
175
200
Number of trees (depth of each tree = 8, max number of features = 4)


Effect of number of max features on bike dataset (Test)
Average RMSE (-ve) from 10 folds CV
−880
−900
−920
−940
−960
−980
−1000
−1020
2
4
6
8
10
Number of max features (depth of each tree = 8, number of trees = 20)


Effect of number of max features on bike dataset (Train)
Average RMSE (-ve) from 10 folds CV
−460
−480
−500
−520
−540
−560
−580
2
4
6
8
10
Number of max features (depth of each tree = 8, number of trees = 20)


Effect of depth of each tree on bike dataset (Test)
Average RMSE (-ve) from 10 folds CV
−900
−1000
−1100
−1200
−1300
−1400
−1500
−1600
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Depth of each tree (max number of features = 4, number of trees = 20)



[263]:
```
gridSuicide_RF = GridSearchCV(pipe_RF, param_grid=param_grid_RF, cv=10,␣
↪n_jobs=-1, verbose=1,
                                  scoring='neg_root_mean_squared_error',␣
↪return_train_score=True).fit(XSuicideCur_F, YSuicide)
```

```
Fitting 10 folds for each of 3800 candidates, totalling 38000 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done 136 tasks      | elapsed:    5.0s
[Parallel(n_jobs=-1)]: Done 386 tasks      | elapsed:   12.3s
[Parallel(n_jobs=-1)]: Done 736 tasks      | elapsed:   21.8s
[Parallel(n_jobs=-1)]: Done 1186 tasks      | elapsed:   36.3s
[Parallel(n_jobs=-1)]: Done 1736 tasks      | elapsed:   54.7s
[Parallel(n_jobs=-1)]: Done 2386 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 3136 tasks      | elapsed:  1.7min
[Parallel(n_jobs=-1)]: Done 3986 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done 4936 tasks      | elapsed:  2.9min
[Parallel(n_jobs=-1)]: Done 5986 tasks      | elapsed:  3.7min
[Parallel(n_jobs=-1)]: Done 7136 tasks      | elapsed:  4.4min
[Parallel(n_jobs=-1)]: Done 8386 tasks      | elapsed:  5.4min
[Parallel(n_jobs=-1)]: Done 9736 tasks      | elapsed:  6.4min
[Parallel(n_jobs=-1)]: Done 11186 tasks      | elapsed:  7.4min
[Parallel(n_jobs=-1)]: Done 12736 tasks      | elapsed:  8.6min
[Parallel(n_jobs=-1)]: Done 14386 tasks      | elapsed:  9.9min
```

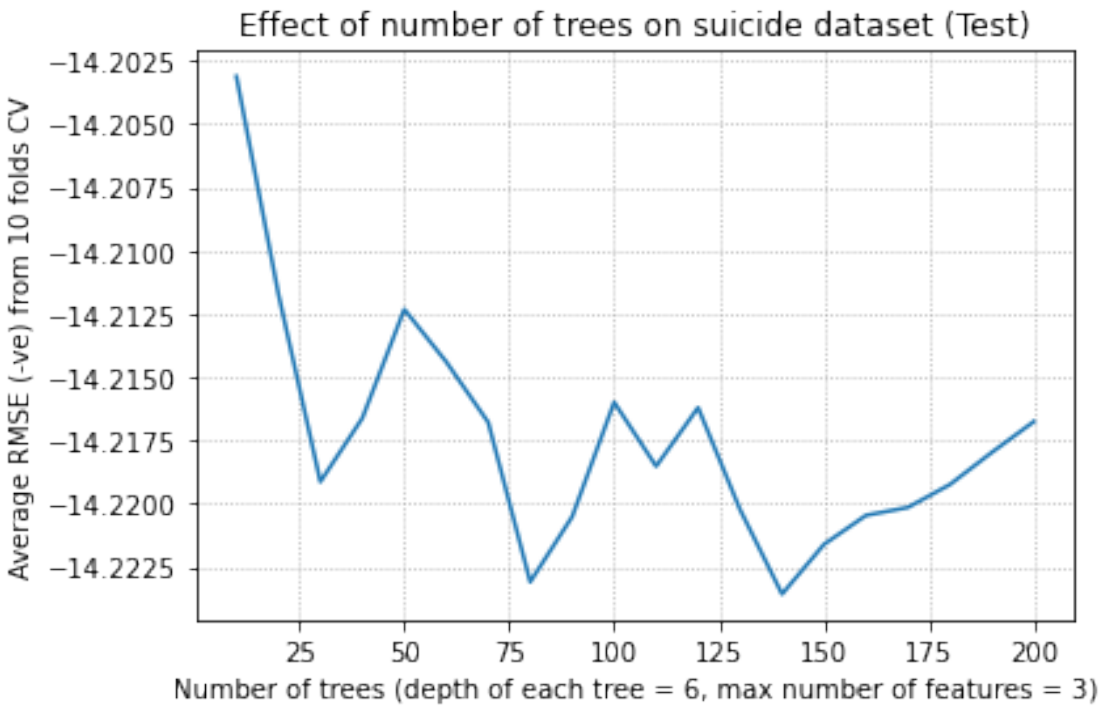
```
[Parallel(n_jobs=-1)]: Done 16136 tasks      | elapsed: 11.4min
[Parallel(n_jobs=-1)]: Done 17986 tasks      | elapsed: 12.9min
[Parallel(n_jobs=-1)]: Done 19936 tasks      | elapsed: 14.4min
[Parallel(n_jobs=-1)]: Done 21986 tasks      | elapsed: 16.0min
[Parallel(n_jobs=-1)]: Done 24136 tasks      | elapsed: 17.7min
[Parallel(n_jobs=-1)]: Done 26386 tasks      | elapsed: 19.4min
[Parallel(n_jobs=-1)]: Done 28736 tasks      | elapsed: 21.2min
[Parallel(n_jobs=-1)]: Done 31186 tasks      | elapsed: 23.1min
[Parallel(n_jobs=-1)]: Done 33736 tasks      | elapsed: 25.2min
[Parallel(n_jobs=-1)]: Done 36386 tasks      | elapsed: 27.3min
[Parallel(n_jobs=-1)]: Done 38000 out of 38000 | elapsed: 28.6min finished
/home/nesl/anaconda3/envs/tflite/lib/python3.8/site-
packages/sklearn/ensemble/_forest.py:832: UserWarning: Some inputs do not have
OOB scores. This probably means too few trees were used to compute any reliable
oob estimates.
  warn("Some inputs do not have OOB scores. "
```

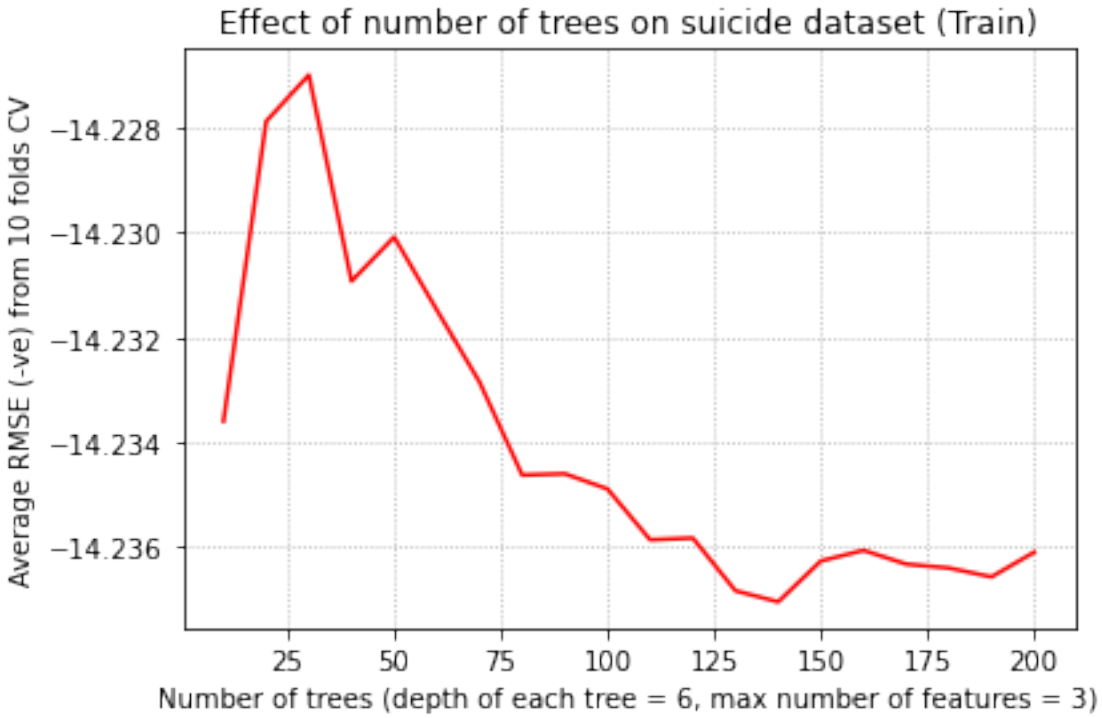
[264]:
```
poly_result = pd.DataFrame(gridSuicide_RF.
 ↪cv_results_)[['mean_test_score','mean_train_score','param_model__max_features','param_model
print('Best parameters (suicide):',gridSuicide_RF.best_params_,',Test RMSE:
 ↪',gridSuicide_RF.best_score_)
print('Train RMSE:',max(poly_result.mean_train_score))
```

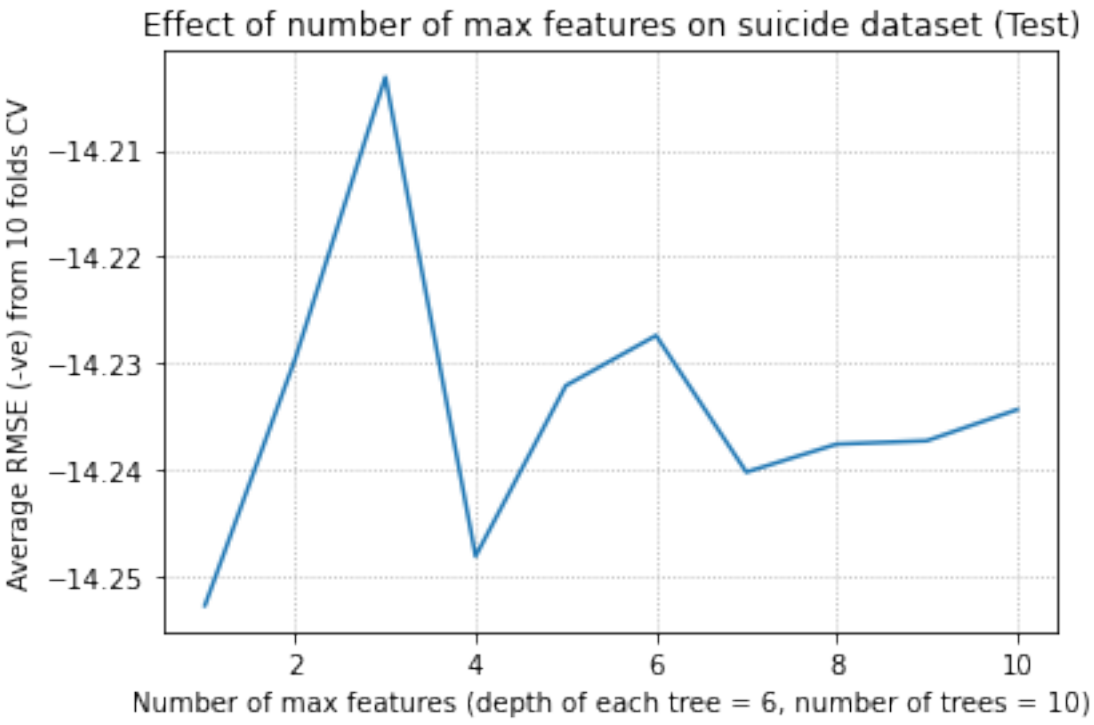
```
Best parameters (suicide): {'model__max_depth': 6, 'model__max_features': 3,
'model__n_estimators': 10} ,Test RMSE: -14.203124085489076
Train RMSE: -14.211886727921348
```

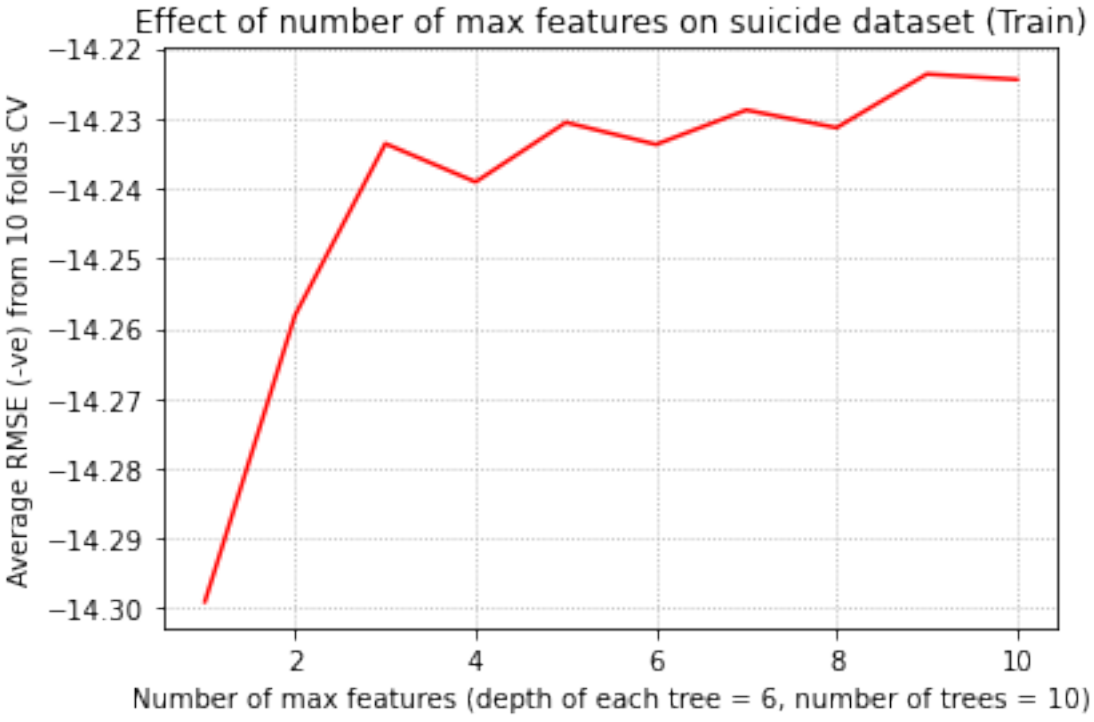
[265]:
```
suicide_score = list((poly_result[(poly_result['param_model__max_depth'] == 6)␣
 ↪& (poly_result['param_model__max_features'] == 3)]).mean_test_score)
suicide_train = list((poly_result[(poly_result['param_model__max_depth'] == 6)␣
 ↪& (poly_result['param_model__max_features'] == 3)]).mean_train_score)
plt.plot(n_estimators,suicide_score)
plt.grid(linestyle=':')
plt.xlabel('Number of trees (depth of each tree = 6, max number of features =␣
 ↪3)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of number of trees on suicide dataset (Test)')
plt.savefig('Q21g.png',dpi=300,bbox_inches='tight')
plt.show()
plt.plot(n_estimators,suicide_train,'r')
plt.grid(linestyle=':')
plt.xlabel('Number of trees (depth of each tree = 6, max number of features =␣
 ↪3)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of number of trees on suicide dataset (Train)')
plt.savefig('Q21h.png',dpi=300,bbox_inches='tight')
plt.show()
```

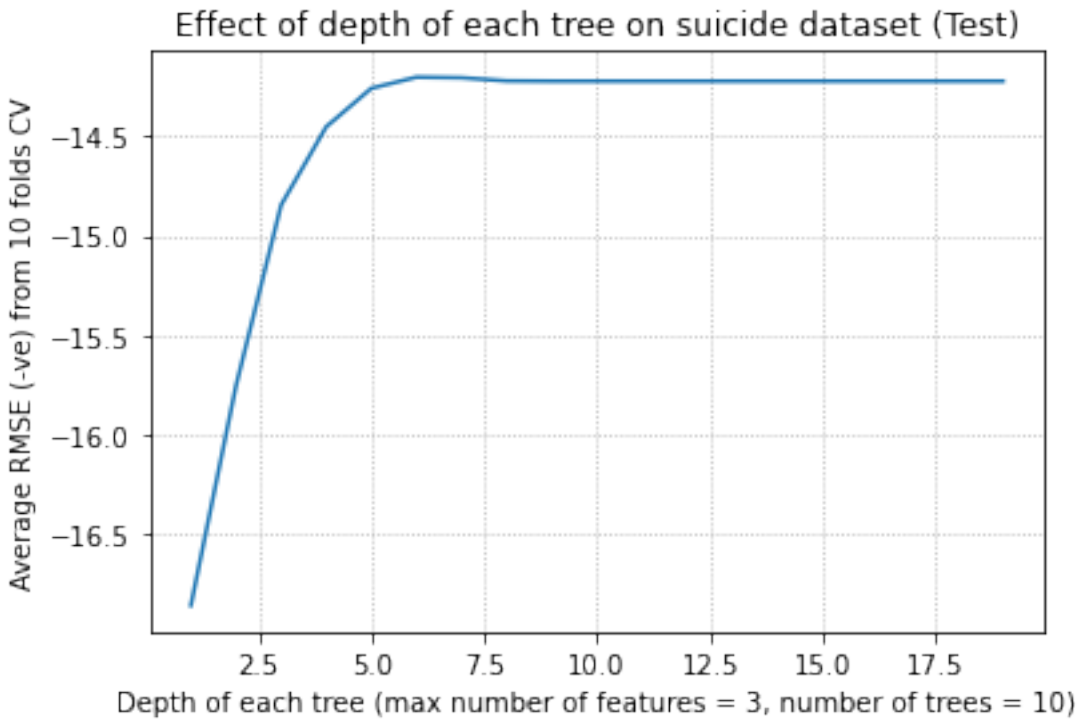
```
suicide_score = list((poly_result[(poly_result['param_model__max_depth'] == 6)␣
 ↪& (poly_result['param_model__n_estimators'] == 10)]).mean_test_score)
suicide_train = list((poly_result[(poly_result['param_model__max_depth'] == 6)␣
 ↪& (poly_result['param_model__n_estimators'] == 10)]).mean_train_score)
plt.plot(max_features,suicide_score)
plt.grid(linestyle=':')
plt.xlabel('Number of max features (depth of each tree = 6, number of trees =␣
 ↪10)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of number of max features on suicide dataset (Test)')
plt.savefig('Q21i.png',dpi=300,bbox_inches='tight')
plt.show()
plt.plot(max_features,suicide_train,'r')
plt.grid(linestyle=':')
plt.xlabel('Number of max features (depth of each tree = 6, number of trees =␣
 ↪10)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of number of max features on suicide dataset (Train)')
plt.savefig('Q21j.png',dpi=300,bbox_inches='tight')
plt.show()

suicide_score = list((poly_result[(poly_result['param_model__max_features'] ==␣
 ↪3) & (poly_result['param_model__n_estimators'] == 10)]).mean_test_score)
suicide_train = list((poly_result[(poly_result['param_model__max_features'] ==␣
 ↪3) & (poly_result['param_model__n_estimators'] == 10)]).mean_train_score)
plt.plot(max_depth,suicide_score)
plt.grid(linestyle=':')
plt.xlabel('Depth of each tree (max number of features = 3, number of trees =␣
 ↪10)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of depth of each tree on suicide dataset (Test)')
plt.savefig('Q21k.png',dpi=300,bbox_inches='tight')
plt.show()
plt.plot(max_depth,suicide_train,'r')
plt.grid(linestyle=':')
plt.xlabel('Depth of each tree  (max number of features = 3, number of trees =␣
 ↪10)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of depth of each tree on suicide dataset (Train)')
plt.savefig('Q21l.png',dpi=300,bbox_inches='tight')
plt.show()
```


Effect of number of trees on suicide dataset (Test)
Average RMSE (-ve) from 10 folds CV
−14.2025
−14.2050
−14.2075
−14.2100
−14.2125
−14.2150
−14.2175
−14.2200
−14.2225
25
50
75
100
125
150
175
200
Number of trees (depth of each tree = 6, max number of features = 3)


Effect of number of trees on suicide dataset (Train)
Average RMSE (-ve) from 10 folds CV
−14.228
−14.230
−14.232
−14.234
−14.236
25
50
75
100
125
150
175
200
Number of trees (depth of each tree = 6, max number of features = 3)


Effect of number of max features on suicide dataset (Test)
Average RMSE (-ve) from 10 folds CV
−14.21
−14.22
−14.23
−14.24
−14.25
2
4
6
8
10
Number of max features (depth of each tree = 6, number of trees = 10)


Effect of number of max features on suicide dataset (Train)
Average RMSE (-ve) from 10 folds CV
−14.22
−14.23
−14.24
−14.25
−14.26
−14.27
−14.28
−14.29
−14.30
2
4
6
8
10
Number of max features (depth of each tree = 6, number of trees = 10)


Effect of depth of each tree on suicide dataset (Test)
Average RMSE (-ve) from 10 folds CV
−14.5
−15.0
−15.5
−16.0
−16.5
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Depth of each tree (max number of features = 3, number of trees = 10)


Effect of depth of each tree on suicide dataset (Train)
Average RMSE (-ve) from 10 folds CV
−14.5
−15.0
−15.5
−16.0
−16.5
−17.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Depth of each tree (max number of features = 3, number of trees = 10)

[267]:
```
param_grid_RF = {
    'model__max_features': np.arange(1,11,1),
    'model__n_estimators': np.arange(10, 50, 10),
    'model__max_depth': np.arange(1, 20, 1)

}
gridTranscode_RF = GridSearchCV(pipe_RF, param_grid=param_grid_RF, cv=10,␣
↪n_jobs=-1, verbose=1,
                                scoring='neg_root_mean_squared_error',␣
↪return_train_score=True).fit(XTranscodeCur_F, YTranscode)
```

```
Fitting 10 folds for each of 760 candidates, totalling 7600 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done 136 tasks      | elapsed:    5.3s
[Parallel(n_jobs=-1)]: Done 386 tasks      | elapsed:   14.5s
[Parallel(n_jobs=-1)]: Done 736 tasks      | elapsed:   31.2s
[Parallel(n_jobs=-1)]: Done 1186 tasks      | elapsed:   57.9s
[Parallel(n_jobs=-1)]: Done 1736 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 2386 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done 3136 tasks      | elapsed:  3.2min
[Parallel(n_jobs=-1)]: Done 3986 tasks      | elapsed:  4.3min
[Parallel(n_jobs=-1)]: Done 4936 tasks      | elapsed:  5.5min
[Parallel(n_jobs=-1)]: Done 5986 tasks      | elapsed:  7.1min
[Parallel(n_jobs=-1)]: Done 7136 tasks      | elapsed:  9.1min
[Parallel(n_jobs=-1)]: Done 7600 out of 7600 | elapsed: 10.1min finished
```

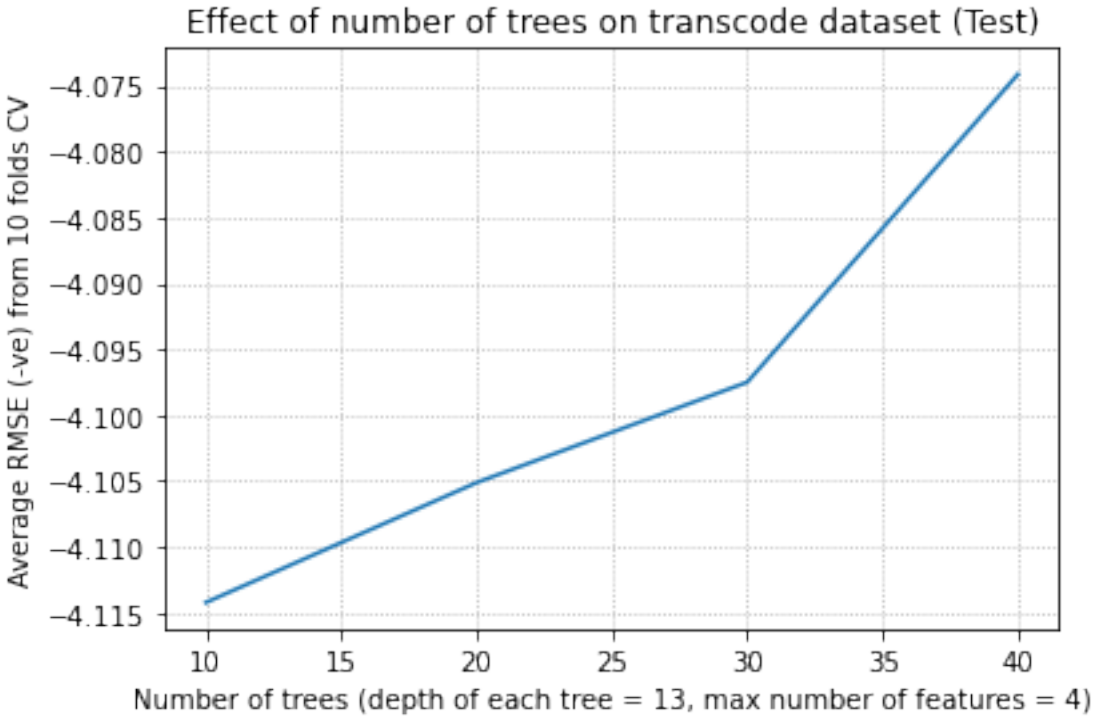
[268]:
```
poly_result = pd.DataFrame(gridTranscode_RF.
↪cv_results_)[['mean_test_score','mean_train_score','param_model__max_features','param_model
print('Best parameters (transcode):',gridTranscode_RF.best_params_,',Test RMSE:
↪',gridTranscode_RF.best_score_)
print('Train RMSE:',max(poly_result.mean_train_score))
```

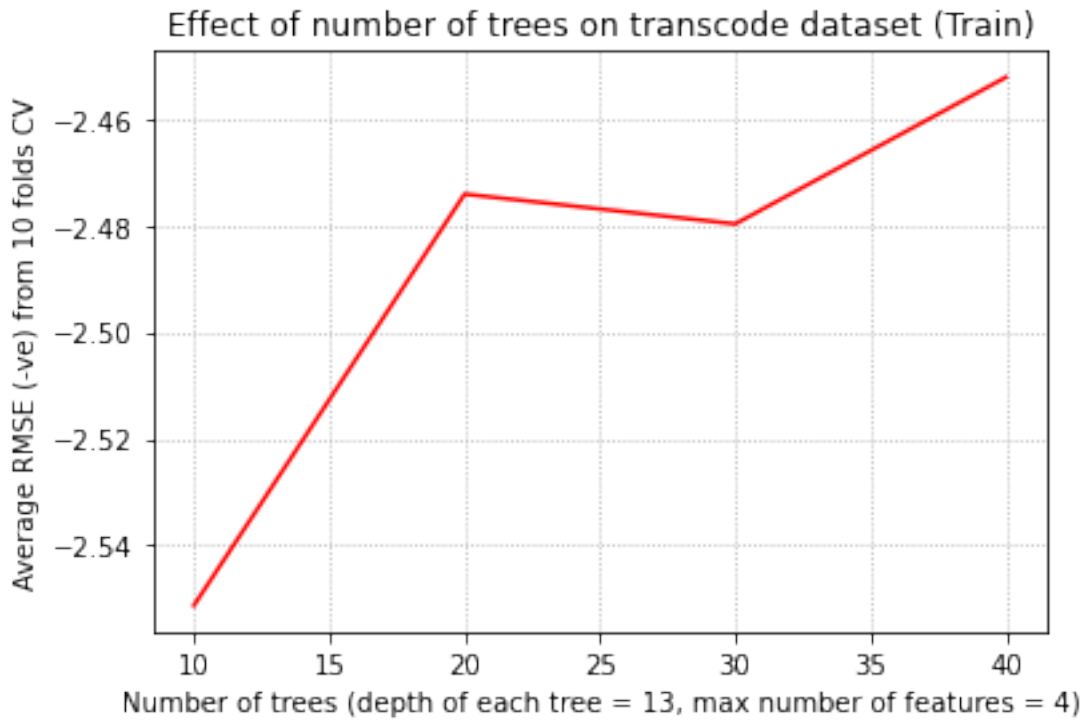
```
Best parameters (transcode): {'model__max_depth': 13, 'model__max_features': 4,
'model__n_estimators': 40} ,Test RMSE: -4.074124938241727
Train RMSE: -0.7462747479626495
```

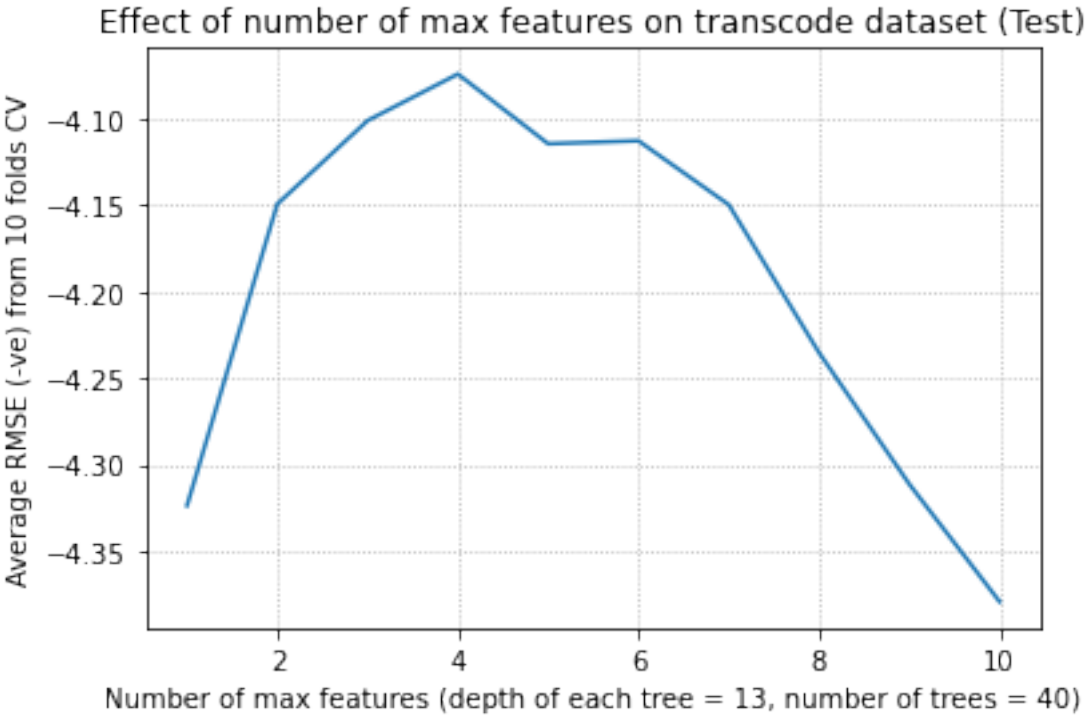
[269]:
```
max_features = np.arange(1,11,1).reshape(10)
n_estimators = np.arange(10, 50, 10).reshape(4)
max_depth = np.arange(1, 20, 1).reshape(19)

transcode_score = list((poly_result[(poly_result['param_model__max_depth'] ==␣
↪13) & (poly_result['param_model__max_features'] == 4)]).mean_test_score)
transcode_train = list((poly_result[(poly_result['param_model__max_depth'] ==␣
↪13) & (poly_result['param_model__max_features'] == 4)]).mean_train_score)
plt.plot(n_estimators,transcode_score)
plt.grid(linestyle=':')
```

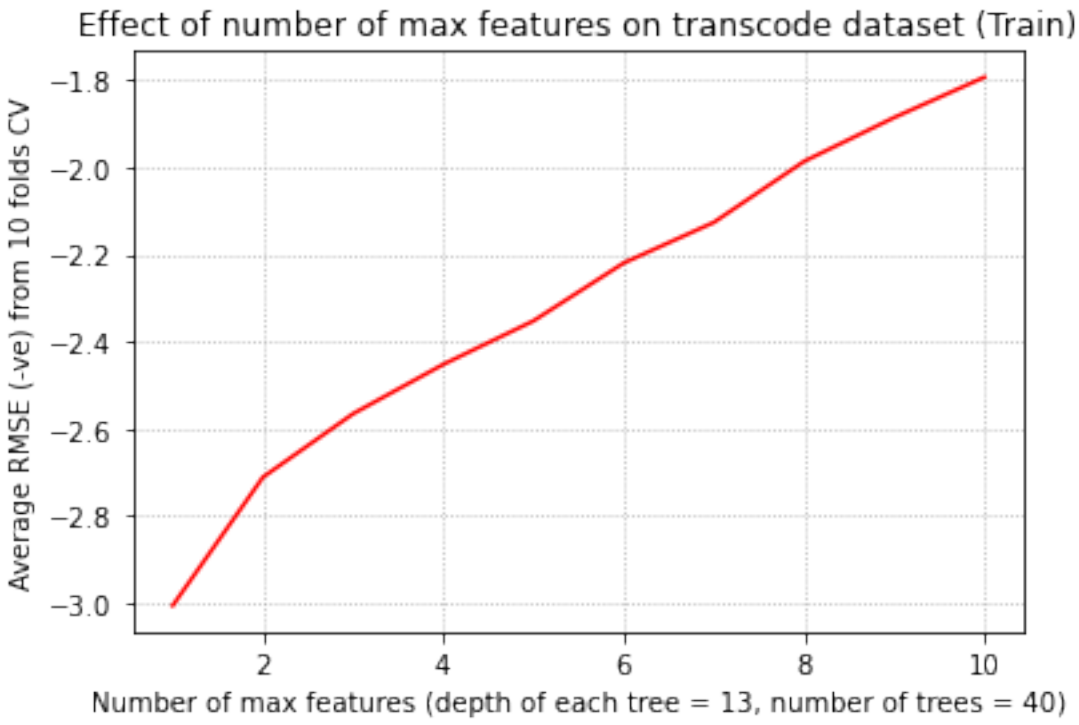
```
plt.xlabel('Number of trees (depth of each tree = 13, max number of features =␣
↪4)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of number of trees on transcode dataset (Test)')
plt.savefig('Q21m.png',dpi=300,bbox_inches='tight')
plt.show()
plt.plot(n_estimators,transcode_train,'r')
plt.grid(linestyle=':')
plt.xlabel('Number of trees (depth of each tree = 13, max number of features =␣
↪4)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of number of trees on transcode dataset (Train)')
plt.savefig('Q21n.png',dpi=300,bbox_inches='tight')
plt.show()

transcode_score = list((poly_result[(poly_result['param_model__max_depth'] ==␣
↪13) & (poly_result['param_model__n_estimators'] == 40)]).mean_test_score)
transcode_train = list((poly_result[(poly_result['param_model__max_depth'] ==␣
↪13) & (poly_result['param_model__n_estimators'] == 40)]).mean_train_score)
plt.plot(max_features,transcode_score)
plt.grid(linestyle=':')
plt.xlabel('Number of max features (depth of each tree = 13, number of trees =␣
↪40)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of number of max features on transcode dataset (Test)')
plt.savefig('Q21o.png',dpi=300,bbox_inches='tight')
plt.show()
plt.plot(max_features,transcode_train,'r')
plt.grid(linestyle=':')
plt.xlabel('Number of max features (depth of each tree = 13, number of trees =␣
↪40)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of number of max features on transcode dataset (Train)')
plt.savefig('Q21p.png',dpi=300,bbox_inches='tight')
plt.show()

transcode_score = list((poly_result[(poly_result['param_model__max_features']␣
↪== 4) & (poly_result['param_model__n_estimators'] == 40)]).mean_test_score)
transcode_train = list((poly_result[(poly_result['param_model__max_features']␣
↪== 4) & (poly_result['param_model__n_estimators'] == 40)]).mean_train_score)
plt.plot(max_depth,transcode_score)
plt.grid(linestyle=':')
plt.xlabel('Depth of each tree (max number of features = 4, number of trees =␣
↪40)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of depth of each tree on transcode dataset (Test)')
```

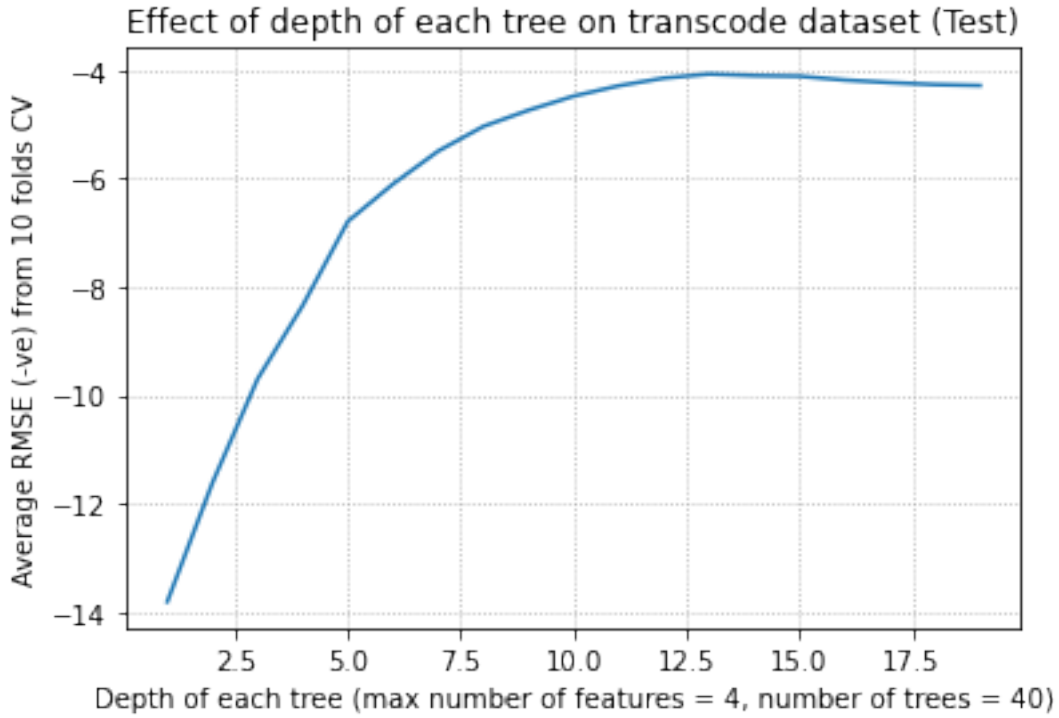
```
plt.savefig('Q21q.png',dpi=300,bbox_inches='tight')
plt.show()
plt.plot(max_depth,transcode_train,'r')
plt.grid(linestyle=':')
plt.xlabel('Depth of each tree  (max number of features = 4, number of trees =␣
↪40)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of depth of each tree on transcode dataset (Train)')
plt.savefig('Q21r.png',dpi=300,bbox_inches='tight')
plt.show()
```

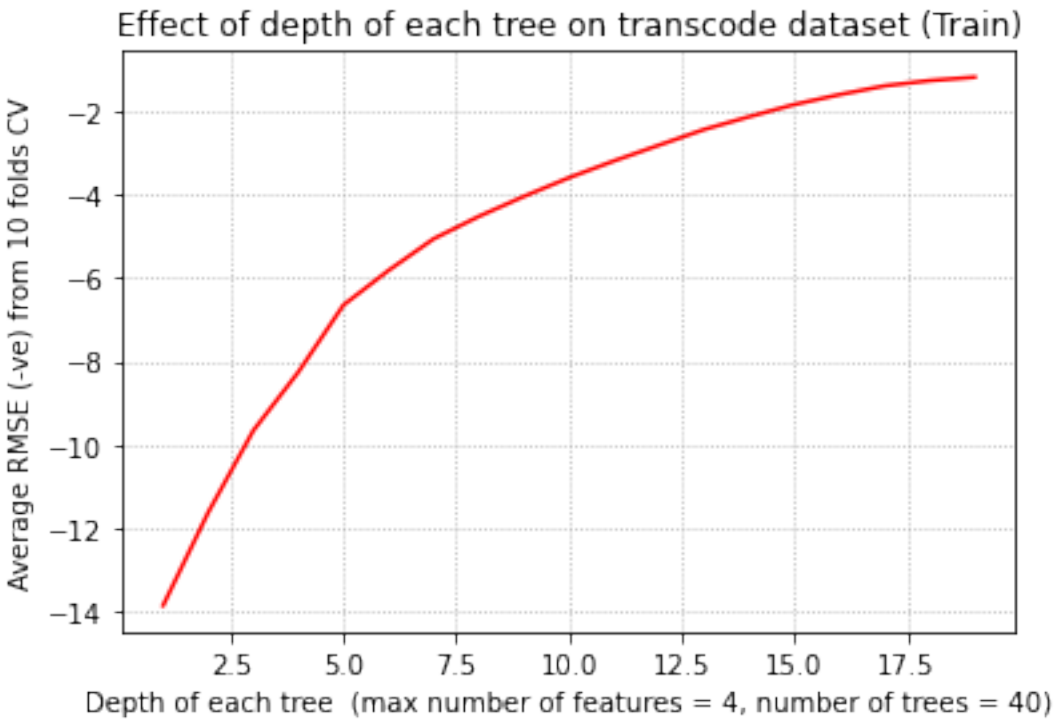



Effect of number of trees on transcode dataset (Train)
Average RMSE (-ve) from 10 folds CV
−2.46
−2.48
−2.50
−2.52
−2.54
10
15
20
25
30
35
40
Number of trees (depth of each tree = 13, max number of features = 4)


Effect of number of max features on transcode dataset (Test)
Average RMSE (-ve) from 10 folds CV
−4.10
−4.15
−4.20
−4.25
−4.30
−4.35
2
4
6
8
10
Number of max features (depth of each tree = 13, number of trees = 40)


Effect of number of max features on transcode dataset (Train)
Average RMSE (-ve) from 10 folds CV
−1.8
−2.0
−2.2
−2.4
−2.6
−2.8
−3.0
2
4
6
8
10
Number of max features (depth of each tree = 13, number of trees = 40)


Effect of depth of each tree on transcode dataset (Test)
Average RMSE (-ve) from 10 folds CV
−4
−6
−8
−10
−12
−14
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Depth of each tree (max number of features = 4, number of trees = 40)



0.11.2 OOB Error for best models

[270]:
```
print('OOB, Bike:',RandomForestRegressor(random_state=42,max_depth=8,
                                          max_features=4, n_estimators=20,␣
 ↪oob_score=True).fit(XBikeCur_F,YBike).oob_score_)
print('OOB, Suicide:',RandomForestRegressor(random_state=42,max_depth=6,␣
 ↪max_features=3, n_estimators=10, oob_score=True).fit(XSuicideCur_F,YSuicide).
 ↪oob_score_)
print('OOB, Transcode:',RandomForestRegressor(random_state=42,max_depth=13,␣
 ↪max_features=4, n_estimators=40, oob_score=True).
 ↪fit(XTranscodeCur_F,YTranscode).oob_score_)
```

```
OOB, Bike: 0.8274957524625527
OOB, Suicide: 0.4220842787293496
```

```
/home/nesl/anaconda3/envs/tflite/lib/python3.8/site-
packages/sklearn/ensemble/_forest.py:832: UserWarning: Some inputs do not have
OOB scores. This probably means too few trees were used to compute any reliable
oob estimates.
  warn("Some inputs do not have OOB scores. "
```

```
OOB, Transcode: 0.9601340475580523
```

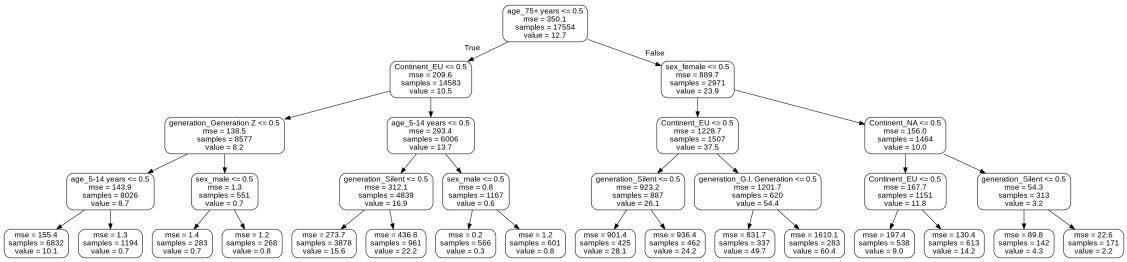
0.11.3 Tree Visualization

[272]:
```
vis_tree = RandomForestRegressor(random_state=42,max_depth=4, max_features=3,␣
↪n_estimators=10).fit(XSuicideCur_F,YSuicide)
```

[281]:
```
chY = SelectKBest(score_func=f_regression, k=10)
XTranscode_Test = chY.fit_transform(suicide_LR.loc[:, suicide_LR.columns !=␣
↪'suicides/100k pop'], suicide_LR["suicides/100k pop"])
column_names = suicide_LR.loc[:, suicide_LR.columns != 'suicides/100k pop'].
↪columns[chY.get_support()]
```

[288]:
```
tree = vis_tree.estimators_[1]
export_graphviz(tree, out_file = 'tree.dot', feature_names = column_names,␣
↪rounded = True, precision = 1)
(graph, ) = pydot.graph_from_dot_file('tree.dot')
Image(graph.create_png())
```

[288]:



[290]:
```
graph.write_png("Q23.png")
```

0.12 Question 24 to 26

[336]:
```
opt = BayesSearchCV(
    lgb.LGBMRegressor(random_state=42,verbose=1,n_jobs=-1),
    {
        'boosting_type': ['gbdt', 'dart','rf'],
        'num_leaves': np.arange(20,1000,10),
        'max_depth': np.arange(1,100,10),
        'n_estimators': np.arange(10,4000,100),
        'reg_alpha': [10.0**x for x in np.arange(-4,4)],
        'reg_lambda': [10.0**x for x in np.arange(-4,4)],
        'subsample': np.arange(0.1,1,0.1),
        'subsample_freq': np.arange(0,50,5),
        'min_split_gain': [10.0**x for x in np.arange(-4,0)]
    },
    n_iter=20,
    cv=10,
    n_jobs=-1,
```

```
    verbose=1,
    random_state=42,
    scoring = 'neg_root_mean_squared_error',
    return_train_score = True
)
```

[337]:
```
_ = opt.fit(XSuicideCur_F,YSuicide)
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    5.4s remaining:   21.6s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    6.3s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   36.7s remaining:  2.4min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   41.2s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   10.0s remaining:   39.9s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   12.4s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   12.8s remaining:   51.2s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   16.4s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   10.8s remaining:   43.0s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   15.6s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    2.7s remaining:   10.7s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    2.8s finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   30.9s remaining:  2.1min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   36.5s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits
```

```
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   27.9s remaining:  1.9min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   36.1s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    7.1s remaining:   28.2s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    8.4s finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   48.5s remaining:  3.2min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   49.6s finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    5.1s remaining:   20.5s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    6.4s finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    6.4s remaining:   25.5s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    8.0s finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   23.8s remaining:  1.6min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   30.7s finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    4.2s remaining:   16.7s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    5.2s finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    9.8s remaining:   39.3s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   12.7s finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    7.5s remaining:   30.0s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   10.0s finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    2.3s remaining:    9.3s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    2.4s finished
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   17.8s remaining:  1.2min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   26.9s finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    1.5s remaining:    6.0s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    1.6s finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   11.5s remaining:   46.2s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   16.9s finished

[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.009148 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 20
[LightGBM] [Info] Number of data points in the train set: 27820, number of used
features: 10
[LightGBM] [Info] Start training from score 12.816097
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

[348]:
```
print('Best parameters (suicide):',opt.best_params_,',Test RMSE:',opt.
 ↪best_score_)
print('Train RMSE:',min(opt.cv_results_['mean_train_score']))
```

```
Best parameters (suicide): OrderedDict([('boosting_type', 'gbdt'), ('max_depth',
81), ('min_split_gain', 0.0001), ('n_estimators', 710), ('num_leaves', 800),
('reg_alpha', 0.1), ('reg_lambda', 0.1), ('subsample', 0.5), ('subsample_freq',
35)]) ,Test RMSE: -14.208668140311447
Train RMSE: -14.277083632830061
```

[365]:
```
optcatLG = BayesSearchCV(
   ␣
 ↪CatBoostRegressor(random_state=42,verbose=1,thread_count=-1,bootstrap_type='Bayesian'),
    {
        'colsample_bylevel': np.arange(0.1,1,0.1),
        'num_trees': np.arange(10,4000,100),
        'l2_leaf_reg': [10.0**x for x in np.arange(-4,4)],
        'num_leaves': np.arange(20,1000,10),
        'max_depth': np.arange(1,16,2),
        'bagging_temperature': np.arange(0.1,10,1),
        'grow_policy': ['Lossguide'],

    },
    n_iter=20,
    cv=10,
    n_jobs=-1,
    verbose=1,
    random_state=42,
    scoring = 'neg_root_mean_squared_error',
    return_train_score = True
)
```

[366]:
```
_ = optcatLG.fit(XSuicideCur_F,YSuicide)
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:  3.7min remaining: 14.7min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  3.8min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:  1.0min remaining:  4.1min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  1.1min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   21.1s remaining:  1.4min

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   24.7s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:  2.3min remaining:  9.1min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  2.5min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:  4.2min remaining: 16.9min

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  4.4min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:  5.2min remaining: 20.8min

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  5.5min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:  4.9min remaining: 19.7min

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  5.2min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:  2.3min remaining:  9.1min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  2.5min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    1.7s remaining:    6.7s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    2.1s finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:  2.5min remaining: 10.0min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  2.6min finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:  1.3min remaining:  5.2min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  1.3min finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:  4.3min remaining: 17.2min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  4.5min finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:  1.7min remaining:  6.9min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  1.9min finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:  5.5min remaining: 21.9min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  5.8min finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:  1.4min remaining:  5.7min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  1.5min finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:  1.3min remaining:  5.4min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  1.5min finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:  3.2min remaining: 12.9min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  3.4min finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   40.0s remaining:  2.7min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   42.4s finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    0.8s remaining:    3.1s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    0.9s finished
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   11.4s remaining:   45.5s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   13.7s finished

0:      learn: 18.7500092       total: 80.7ms   remaining: 16.9s
1:      learn: 18.5363022       total: 109ms    remaining: 11.3s
2:      learn: 18.3454296       total: 131ms    remaining: 9.03s
3:      learn: 18.1596398       total: 148ms    remaining: 7.63s
4:      learn: 17.9805087       total: 168ms    remaining: 6.88s
5:      learn: 17.8066175       total: 189ms    remaining: 6.43s
6:      learn: 17.6381880       total: 208ms    remaining: 6.03s
7:      learn: 17.4742820       total: 224ms    remaining: 5.66s
8:      learn: 17.3306908       total: 241ms    remaining: 5.38s
9:      learn: 17.1861840       total: 260ms    remaining: 5.2s
10:     learn: 17.0498120       total: 272ms    remaining: 4.92s
11:     learn: 16.9165353       total: 287ms    remaining: 4.73s
12:     learn: 16.7924394       total: 297ms    remaining: 4.51s
13:     learn: 16.6769927       total: 305ms    remaining: 4.27s
14:     learn: 16.5609446       total: 313ms    remaining: 4.06s
15:     learn: 16.4524829       total: 320ms    remaining: 3.87s
16:     learn: 16.3477251       total: 327ms    remaining: 3.71s
17:     learn: 16.2446847       total: 333ms    remaining: 3.55s
18:     learn: 16.1490823       total: 340ms    remaining: 3.42s
19:     learn: 16.0585175       total: 347ms    remaining: 3.3s
20:     learn: 15.9727177       total: 359ms    remaining: 3.23s
21:     learn: 15.8890575       total: 367ms    remaining: 3.14s
22:     learn: 15.8106286       total: 375ms    remaining: 3.05s
23:     learn: 15.7362430       total: 382ms    remaining: 2.96s
24:     learn: 15.6629359       total: 388ms    remaining: 2.87s
25:     learn: 15.5961167       total: 397ms    remaining: 2.81s
26:     learn: 15.5311558       total: 404ms    remaining: 2.74s
27:     learn: 15.4672843       total: 412ms    remaining: 2.68s
28:     learn: 15.4082479       total: 420ms    remaining: 2.62s
29:     learn: 15.3519236       total: 428ms    remaining: 2.56s
30:     learn: 15.3003674       total: 434ms    remaining: 2.5s
31:     learn: 15.2497666       total: 445ms    remaining: 2.48s
32:     learn: 15.2028897       total: 460ms    remaining: 2.47s
33:     learn: 15.1562544       total: 474ms    remaining: 2.45s
34:     learn: 15.1138619       total: 494ms    remaining: 2.47s
35:     learn: 15.0733027       total: 507ms    remaining: 2.45s
36:     learn: 15.0355188       total: 516ms    remaining: 2.41s
37:     learn: 14.9983799       total: 524ms    remaining: 2.37s
38:     learn: 14.9625808       total: 532ms    remaining: 2.33s
39:     learn: 14.9283259       total: 542ms    remaining: 2.3s
40:     learn: 14.8944741       total: 549ms    remaining: 2.26s
41:     learn: 14.8651852       total: 556ms    remaining: 2.22s
42:     learn: 14.8378457       total: 562ms    remaining: 2.18s
```

```
43:     learn: 14.8092272       total: 569ms    remaining: 2.15s
44:     learn: 14.7830039       total: 576ms    remaining: 2.11s
45:     learn: 14.7584260       total: 583ms    remaining: 2.08s
46:     learn: 14.7344036       total: 590ms    remaining: 2.04s
47:     learn: 14.7099635       total: 597ms    remaining: 2.01s
48:     learn: 14.6890615       total: 603ms    remaining: 1.98s
49:     learn: 14.6677257       total: 613ms    remaining: 1.96s
50:     learn: 14.6473459       total: 619ms    remaining: 1.93s
51:     learn: 14.6284038       total: 625ms    remaining: 1.9s
52:     learn: 14.6109691       total: 632ms    remaining: 1.87s
53:     learn: 14.5932850       total: 640ms    remaining: 1.85s
54:     learn: 14.5769437       total: 649ms    remaining: 1.83s
55:     learn: 14.5607904       total: 661ms    remaining: 1.82s
56:     learn: 14.5458297       total: 678ms    remaining: 1.82s
57:     learn: 14.5316360       total: 691ms    remaining: 1.81s
58:     learn: 14.5178944       total: 710ms    remaining: 1.82s
59:     learn: 14.5054327       total: 725ms    remaining: 1.81s
60:     learn: 14.4935072       total: 742ms    remaining: 1.81s
61:     learn: 14.4808768       total: 756ms    remaining: 1.8s
62:     learn: 14.4703134       total: 764ms    remaining: 1.78s
63:     learn: 14.4590760       total: 772ms    remaining: 1.76s
64:     learn: 14.4485320       total: 780ms    remaining: 1.74s
65:     learn: 14.4383553       total: 787ms    remaining: 1.72s
66:     learn: 14.4305501       total: 793ms    remaining: 1.69s
67:     learn: 14.4228169       total: 800ms    remaining: 1.67s
68:     learn: 14.4136477       total: 808ms    remaining: 1.65s
69:     learn: 14.4055441       total: 814ms    remaining: 1.63s
70:     learn: 14.3978645       total: 821ms    remaining: 1.61s
71:     learn: 14.3904773       total: 828ms    remaining: 1.59s
72:     learn: 14.3831145       total: 835ms    remaining: 1.57s
73:     learn: 14.3766183       total: 848ms    remaining: 1.56s
74:     learn: 14.3702829       total: 865ms    remaining: 1.56s
75:     learn: 14.3644747       total: 875ms    remaining: 1.54s
76:     learn: 14.3586870       total: 882ms    remaining: 1.52s
77:     learn: 14.3531385       total: 890ms    remaining: 1.51s
78:     learn: 14.3480683       total: 899ms    remaining: 1.49s
79:     learn: 14.3432232       total: 910ms    remaining: 1.48s
80:     learn: 14.3388484       total: 920ms    remaining: 1.47s
81:     learn: 14.3347025       total: 927ms    remaining: 1.45s
82:     learn: 14.3306326       total: 934ms    remaining: 1.43s
83:     learn: 14.3262494       total: 941ms    remaining: 1.41s
84:     learn: 14.3226922       total: 957ms    remaining: 1.41s
85:     learn: 14.3190772       total: 973ms    remaining: 1.4s
86:     learn: 14.3162526       total: 984ms    remaining: 1.39s
87:     learn: 14.3132440       total: 994ms    remaining: 1.38s
88:     learn: 14.3095261       total: 1s       remaining: 1.36s
89:     learn: 14.3060872       total: 1.01s    remaining: 1.34s
90:     learn: 14.3029275       total: 1.02s    remaining: 1.33s
```

```
91:     learn: 14.2998731       total: 1.02s    remaining: 1.31s
92:     learn: 14.2971486       total: 1.04s    remaining: 1.3s
93:     learn: 14.2943956       total: 1.05s    remaining: 1.3s
94:     learn: 14.2919495       total: 1.06s    remaining: 1.28s
95:     learn: 14.2895856       total: 1.07s    remaining: 1.27s
96:     learn: 14.2872062       total: 1.07s    remaining: 1.25s
97:     learn: 14.2854278       total: 1.08s    remaining: 1.24s
98:     learn: 14.2835600       total: 1.09s    remaining: 1.22s
99:     learn: 14.2817032       total: 1.09s    remaining: 1.2s
100:    learn: 14.2796304       total: 1.1s     remaining: 1.19s
101:    learn: 14.2779050       total: 1.11s    remaining: 1.18s
102:    learn: 14.2760406       total: 1.12s    remaining: 1.17s
103:    learn: 14.2752676       total: 1.13s    remaining: 1.15s
104:    learn: 14.2738095       total: 1.15s    remaining: 1.15s
105:    learn: 14.2722242       total: 1.16s    remaining: 1.14s
106:    learn: 14.2710010       total: 1.18s    remaining: 1.13s
107:    learn: 14.2703889       total: 1.19s    remaining: 1.12s
108:    learn: 14.2688643       total: 1.2s     remaining: 1.11s
109:    learn: 14.2676158       total: 1.22s    remaining: 1.11s
110:    learn: 14.2664812       total: 1.23s    remaining: 1.1s
111:    learn: 14.2653041       total: 1.25s    remaining: 1.09s
112:    learn: 14.2640833       total: 1.25s    remaining: 1.08s
113:    learn: 14.2629419       total: 1.26s    remaining: 1.06s
114:    learn: 14.2620105       total: 1.27s    remaining: 1.05s
115:    learn: 14.2609795       total: 1.27s    remaining: 1.03s
116:    learn: 14.2598936       total: 1.28s    remaining: 1.02s
117:    learn: 14.2588922       total: 1.29s    remaining: 1s
118:    learn: 14.2578370       total: 1.3s     remaining: 991ms
119:    learn: 14.2569778       total: 1.3s     remaining: 978ms
120:    learn: 14.2560519       total: 1.31s    remaining: 964ms
121:    learn: 14.2552435       total: 1.32s    remaining: 950ms
122:    learn: 14.2544578       total: 1.32s    remaining: 936ms
123:    learn: 14.2537199       total: 1.33s    remaining: 921ms
124:    learn: 14.2530382       total: 1.34s    remaining: 909ms
125:    learn: 14.2522859       total: 1.34s    remaining: 894ms
126:    learn: 14.2518264       total: 1.35s    remaining: 880ms
127:    learn: 14.2514307       total: 1.35s    remaining: 869ms
128:    learn: 14.2508261       total: 1.36s    remaining: 856ms
129:    learn: 14.2503406       total: 1.37s    remaining: 843ms
130:    learn: 14.2497344       total: 1.38s    remaining: 829ms
131:    learn: 14.2491666       total: 1.38s    remaining: 816ms
132:    learn: 14.2486610       total: 1.39s    remaining: 804ms
133:    learn: 14.2481656       total: 1.4s     remaining: 791ms
134:    learn: 14.2476088       total: 1.4s     remaining: 779ms
135:    learn: 14.2471341       total: 1.41s    remaining: 767ms
136:    learn: 14.2467436       total: 1.42s    remaining: 754ms
137:    learn: 14.2464863       total: 1.42s    remaining: 741ms
138:    learn: 14.2460054       total: 1.43s    remaining: 730ms
```

```
139:    learn: 14.2455564       total: 1.44s    remaining: 719ms
140:    learn: 14.2453117       total: 1.45s    remaining: 708ms
141:    learn: 14.2449015       total: 1.46s    remaining: 697ms
142:    learn: 14.2444930       total: 1.46s    remaining: 685ms
143:    learn: 14.2442160       total: 1.47s    remaining: 673ms
144:    learn: 14.2439394       total: 1.47s    remaining: 661ms
145:    learn: 14.2436378       total: 1.48s    remaining: 649ms
146:    learn: 14.2433377       total: 1.49s    remaining: 637ms
147:    learn: 14.2429854       total: 1.49s    remaining: 625ms
148:    learn: 14.2428201       total: 1.5s     remaining: 615ms
149:    learn: 14.2425693       total: 1.51s    remaining: 605ms
150:    learn: 14.2423171       total: 1.52s    remaining: 593ms
151:    learn: 14.2420652       total: 1.52s    remaining: 581ms
152:    learn: 14.2418592       total: 1.53s    remaining: 570ms
153:    learn: 14.2416816       total: 1.54s    remaining: 561ms
154:    learn: 14.2414019       total: 1.56s    remaining: 554ms
155:    learn: 14.2411825       total: 1.57s    remaining: 544ms
156:    learn: 14.2409665       total: 1.58s    remaining: 533ms
157:    learn: 14.2406676       total: 1.59s    remaining: 522ms
158:    learn: 14.2404625       total: 1.59s    remaining: 511ms
159:    learn: 14.2402725       total: 1.6s     remaining: 500ms
160:    learn: 14.2401220       total: 1.61s    remaining: 489ms
161:    learn: 14.2399349       total: 1.61s    remaining: 478ms
162:    learn: 14.2397785       total: 1.62s    remaining: 467ms
163:    learn: 14.2396328       total: 1.63s    remaining: 456ms
164:    learn: 14.2394435       total: 1.63s    remaining: 445ms
165:    learn: 14.2392748       total: 1.64s    remaining: 436ms
166:    learn: 14.2391025       total: 1.66s    remaining: 428ms
167:    learn: 14.2389716       total: 1.68s    remaining: 420ms
168:    learn: 14.2388293       total: 1.7s     remaining: 412ms
169:    learn: 14.2387195       total: 1.71s    remaining: 402ms
170:    learn: 14.2386099       total: 1.72s    remaining: 393ms
171:    learn: 14.2384334       total: 1.74s    remaining: 385ms
172:    learn: 14.2383986       total: 1.75s    remaining: 374ms
173:    learn: 14.2382953       total: 1.76s    remaining: 364ms
174:    learn: 14.2381849       total: 1.77s    remaining: 354ms
175:    learn: 14.2380440       total: 1.77s    remaining: 343ms
176:    learn: 14.2379349       total: 1.78s    remaining: 332ms
177:    learn: 14.2378349       total: 1.79s    remaining: 322ms
178:    learn: 14.2377186       total: 1.8s     remaining: 311ms
179:    learn: 14.2376157       total: 1.8s     remaining: 300ms
180:    learn: 14.2375169       total: 1.81s    remaining: 290ms
181:    learn: 14.2374048       total: 1.82s    remaining: 280ms
182:    learn: 14.2373807       total: 1.82s    remaining: 269ms
183:    learn: 14.2372946       total: 1.83s    remaining: 258ms
184:    learn: 14.2371619       total: 1.84s    remaining: 248ms
185:    learn: 14.2371105       total: 1.85s    remaining: 238ms
186:    learn: 14.2370025       total: 1.85s    remaining: 228ms
```

```
187:    learn: 14.2369041       total: 1.86s    remaining: 218ms
188:    learn: 14.2368190       total: 1.87s    remaining: 207ms
189:    learn: 14.2366770       total: 1.87s    remaining: 197ms
190:    learn: 14.2365976       total: 1.88s    remaining: 187ms
191:    learn: 14.2364866       total: 1.89s    remaining: 177ms
192:    learn: 14.2364431       total: 1.89s    remaining: 167ms
193:    learn: 14.2363738       total: 1.9s     remaining: 157ms
194:    learn: 14.2362309       total: 1.91s    remaining: 147ms
195:    learn: 14.2361338       total: 1.92s    remaining: 137ms
196:    learn: 14.2360710       total: 1.93s    remaining: 127ms
197:    learn: 14.2360518       total: 1.93s    remaining: 117ms
198:    learn: 14.2359952       total: 1.94s    remaining: 107ms
199:    learn: 14.2359384       total: 1.96s    remaining: 97.9ms
200:    learn: 14.2358970       total: 1.97s    remaining: 88.4ms
201:    learn: 14.2358969       total: 1.98s    remaining: 78.3ms
202:    learn: 14.2358341       total: 1.99s    remaining: 68.7ms
203:    learn: 14.2357872       total: 2s       remaining: 58.9ms
204:    learn: 14.2357431       total: 2.01s    remaining: 49.1ms
205:    learn: 14.2356997       total: 2.03s    remaining: 39.5ms
206:    learn: 14.2356604       total: 2.05s    remaining: 29.7ms
207:    learn: 14.2356154       total: 2.07s    remaining: 19.9ms
208:    learn: 14.2355728       total: 2.09s    remaining: 10ms
209:    learn: 14.2355311       total: 2.1s     remaining: 0us
```

[367]:
```
print('Best parameters (suicide):',optcatLG.best_params_,',Test RMSE:',optcatLG.
 ↪best_score_)
print('Train RMSE:',min(optcatLG.cv_results_['mean_train_score']))
```

```
Best parameters (suicide): OrderedDict([('bagging_temperature', 2.1),
('colsample_bylevel', 0.8), ('grow_policy', 'Lossguide'), ('l2_leaf_reg',
100.0), ('max_depth', 5), ('num_leaves', 280), ('num_trees', 210)]) ,Test RMSE:
-14.221058625294384
Train RMSE: -17.120537619115954
```

[371]:
```
optcat = BayesSearchCV(
    ␣
 ↪CatBoostRegressor(random_state=42,verbose=1,thread_count=-1,bootstrap_type='Bayesian',used_
    {
        'colsample_bylevel': np.arange(0.1,1,0.1),
        'num_trees': np.arange(10,1000,100),
        'l2_leaf_reg': [10.0**x for x in np.arange(-4,4)],
        'max_depth': np.arange(1,16,2),
        'bagging_temperature': np.arange(0.1,10,1),
        'grow_policy': ['SymmetricTree','Depthwise'],
        'score_function': ['Cosine','L2']

    },
```

```
    n_iter=10,
    cv=10,
    n_jobs=-1,
    verbose=1,
    random_state=42,
    scoring = 'neg_root_mean_squared_error',
    return_train_score = True
)
```

[372]:
```
_ = optcat.fit(XSuicideCur_F,YSuicide)
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   19.0s remaining:  1.3min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   19.9s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   13.2s remaining:   52.9s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   17.2s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   11.5s remaining:   46.1s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   13.3s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    1.1s remaining:    4.3s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    2.5s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   19.2s remaining:  1.3min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   20.8s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   39.0s remaining:  2.6min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   44.0s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:   51.3s remaining:  3.4min
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   56.5s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    0.4s remaining:    1.6s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    0.5s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    4.4s remaining:   17.8s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    6.2s finished

Fitting 10 folds for each of 1 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  10 | elapsed:    4.0s remaining:   15.9s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    5.0s finished

0:      learn: 18.7191733       total: 27.4ms   remaining: 5.74s
1:      learn: 18.4847287       total: 46.3ms   remaining: 4.82s
2:      learn: 18.2691763       total: 59.8ms   remaining: 4.12s
3:      learn: 18.0598299       total: 70.7ms   remaining: 3.64s
4:      learn: 17.8606153       total: 80.9ms   remaining: 3.32s
5:      learn: 17.6673255       total: 93.3ms   remaining: 3.17s
6:      learn: 17.4964795       total: 104ms    remaining: 3.01s
7:      learn: 17.3252520       total: 115ms    remaining: 2.91s
8:      learn: 17.1618552       total: 127ms    remaining: 2.84s
9:      learn: 17.0137132       total: 138ms    remaining: 2.76s
10:     learn: 16.8731387       total: 149ms    remaining: 2.69s
11:     learn: 16.7290527       total: 161ms    remaining: 2.66s
12:     learn: 16.5923386       total: 175ms    remaining: 2.64s
13:     learn: 16.4626685       total: 187ms    remaining: 2.61s
14:     learn: 16.3457840       total: 197ms    remaining: 2.56s
15:     learn: 16.2343312       total: 204ms    remaining: 2.48s
16:     learn: 16.1287620       total: 211ms    remaining: 2.4s
17:     learn: 16.0331878       total: 217ms    remaining: 2.32s
18:     learn: 15.9414167       total: 226ms    remaining: 2.27s
19:     learn: 15.8488464       total: 238ms    remaining: 2.26s
20:     learn: 15.7641405       total: 249ms    remaining: 2.24s
21:     learn: 15.6811181       total: 257ms    remaining: 2.2s
22:     learn: 15.6018050       total: 266ms    remaining: 2.16s
23:     learn: 15.5295769       total: 274ms    remaining: 2.12s
24:     learn: 15.4615600       total: 278ms    remaining: 2.06s
25:     learn: 15.4010054       total: 281ms    remaining: 1.99s
26:     learn: 15.3347935       total: 285ms    remaining: 1.93s
27:     learn: 15.2765963       total: 288ms    remaining: 1.87s
28:     learn: 15.2235775       total: 292ms    remaining: 1.82s
29:     learn: 15.1696349       total: 295ms    remaining: 1.77s
30:     learn: 15.1176547       total: 299ms    remaining: 1.73s
31:     learn: 15.0705411       total: 303ms    remaining: 1.68s
32:     learn: 15.0291385       total: 306ms    remaining: 1.64s
33:     learn: 14.9851488       total: 309ms    remaining: 1.6s
```

```
34:     learn: 14.9482918       total: 312ms    remaining: 1.56s
35:     learn: 14.9113644       total: 315ms    remaining: 1.52s
36:     learn: 14.8724156       total: 319ms    remaining: 1.49s
37:     learn: 14.8392572       total: 322ms    remaining: 1.46s
38:     learn: 14.8043998       total: 327ms    remaining: 1.43s
39:     learn: 14.7726276       total: 331ms    remaining: 1.41s
40:     learn: 14.7420897       total: 335ms    remaining: 1.38s
41:     learn: 14.7132986       total: 339ms    remaining: 1.36s
42:     learn: 14.6875853       total: 343ms    remaining: 1.33s
43:     learn: 14.6618623       total: 349ms    remaining: 1.31s
44:     learn: 14.6368302       total: 359ms    remaining: 1.32s
45:     learn: 14.6132383       total: 370ms    remaining: 1.32s
46:     learn: 14.5942074       total: 373ms    remaining: 1.29s
47:     learn: 14.5780472       total: 376ms    remaining: 1.27s
48:     learn: 14.5590305       total: 380ms    remaining: 1.25s
49:     learn: 14.5435092       total: 383ms    remaining: 1.22s
50:     learn: 14.5291498       total: 385ms    remaining: 1.2s
51:     learn: 14.5119938       total: 389ms    remaining: 1.18s
52:     learn: 14.4961801       total: 393ms    remaining: 1.17s
53:     learn: 14.4810523       total: 397ms    remaining: 1.15s
54:     learn: 14.4688989       total: 400ms    remaining: 1.13s
55:     learn: 14.4564797       total: 405ms    remaining: 1.11s
56:     learn: 14.4442092       total: 408ms    remaining: 1.09s
57:     learn: 14.4341810       total: 411ms    remaining: 1.08s
58:     learn: 14.4241614       total: 414ms    remaining: 1.06s
59:     learn: 14.4139313       total: 418ms    remaining: 1.04s
60:     learn: 14.4042992       total: 422ms    remaining: 1.03s
61:     learn: 14.3944398       total: 433ms    remaining: 1.03s
62:     learn: 14.3859013       total: 436ms    remaining: 1.02s
63:     learn: 14.3794900       total: 439ms    remaining: 1s
64:     learn: 14.3718482       total: 443ms    remaining: 989ms
65:     learn: 14.3642527       total: 447ms    remaining: 975ms
66:     learn: 14.3575394       total: 456ms    remaining: 972ms
67:     learn: 14.3511975       total: 465ms    remaining: 970ms
68:     learn: 14.3450326       total: 468ms    remaining: 957ms
69:     learn: 14.3387742       total: 474ms    remaining: 947ms
70:     learn: 14.3330650       total: 478ms    remaining: 935ms
71:     learn: 14.3282933       total: 481ms    remaining: 922ms
72:     learn: 14.3234411       total: 484ms    remaining: 909ms
73:     learn: 14.3190991       total: 487ms    remaining: 896ms
74:     learn: 14.3144274       total: 491ms    remaining: 884ms
75:     learn: 14.3102022       total: 495ms    remaining: 872ms
76:     learn: 14.3075540       total: 497ms    remaining: 859ms
77:     learn: 14.3040805       total: 500ms    remaining: 846ms
78:     learn: 14.3019148       total: 503ms    remaining: 833ms
79:     learn: 14.2986455       total: 506ms    remaining: 822ms
80:     learn: 14.2950141       total: 510ms    remaining: 812ms
81:     learn: 14.2913209       total: 515ms    remaining: 804ms
```

```
82:     learn: 14.2884877       total: 518ms    remaining: 793ms
83:     learn: 14.2851761       total: 524ms    remaining: 786ms
84:     learn: 14.2822791       total: 528ms    remaining: 777ms
85:     learn: 14.2796522       total: 533ms    remaining: 768ms
86:     learn: 14.2784887       total: 535ms    remaining: 756ms
87:     learn: 14.2760853       total: 539ms    remaining: 747ms
88:     learn: 14.2736147       total: 543ms    remaining: 738ms
89:     learn: 14.2718784       total: 546ms    remaining: 728ms
90:     learn: 14.2697911       total: 550ms    remaining: 719ms
91:     learn: 14.2683446       total: 553ms    remaining: 709ms
92:     learn: 14.2666183       total: 564ms    remaining: 710ms
93:     learn: 14.2647377       total: 576ms    remaining: 710ms
94:     learn: 14.2634273       total: 585ms    remaining: 708ms
95:     learn: 14.2622567       total: 595ms    remaining: 706ms
96:     learn: 14.2606360       total: 607ms    remaining: 707ms
97:     learn: 14.2596810       total: 614ms    remaining: 702ms
98:     learn: 14.2584780       total: 632ms    remaining: 709ms
99:     learn: 14.2570643       total: 646ms    remaining: 711ms
100:    learn: 14.2563976       total: 654ms    remaining: 706ms
101:    learn: 14.2550585       total: 666ms    remaining: 706ms
102:    learn: 14.2539135       total: 677ms    remaining: 704ms
103:    learn: 14.2530299       total: 689ms    remaining: 702ms
104:    learn: 14.2520912       total: 698ms    remaining: 698ms
105:    learn: 14.2515918       total: 706ms    remaining: 693ms
106:    learn: 14.2507956       total: 715ms    remaining: 688ms
107:    learn: 14.2497105       total: 722ms    remaining: 682ms
108:    learn: 14.2489917       total: 725ms    remaining: 672ms
109:    learn: 14.2481695       total: 729ms    remaining: 662ms
110:    learn: 14.2474686       total: 732ms    remaining: 653ms
111:    learn: 14.2473389       total: 734ms    remaining: 642ms
112:    learn: 14.2466822       total: 737ms    remaining: 633ms
113:    learn: 14.2461193       total: 741ms    remaining: 624ms
114:    learn: 14.2455175       total: 745ms    remaining: 616ms
115:    learn: 14.2451783       total: 748ms    remaining: 606ms
116:    learn: 14.2446000       total: 751ms    remaining: 597ms
117:    learn: 14.2440773       total: 754ms    remaining: 588ms
118:    learn: 14.2438123       total: 760ms    remaining: 582ms
119:    learn: 14.2431350       total: 767ms    remaining: 576ms
120:    learn: 14.2426885       total: 771ms    remaining: 567ms
121:    learn: 14.2424860       total: 773ms    remaining: 558ms
122:    learn: 14.2422998       total: 776ms    remaining: 549ms
123:    learn: 14.2420018       total: 779ms    remaining: 540ms
124:    learn: 14.2417081       total: 782ms    remaining: 532ms
125:    learn: 14.2412720       total: 786ms    remaining: 524ms
126:    learn: 14.2409185       total: 789ms    remaining: 516ms
127:    learn: 14.2408056       total: 791ms    remaining: 507ms
128:    learn: 14.2406258       total: 794ms    remaining: 498ms
129:    learn: 14.2401779       total: 798ms    remaining: 491ms
```

```
130:    learn: 14.2398349       total: 802ms    remaining: 484ms
131:    learn: 14.2396183       total: 804ms    remaining: 475ms
132:    learn: 14.2395230       total: 807ms    remaining: 467ms
133:    learn: 14.2393270       total: 810ms    remaining: 459ms
134:    learn: 14.2391028       total: 813ms    remaining: 452ms
135:    learn: 14.2388486       total: 816ms    remaining: 444ms
136:    learn: 14.2386485       total: 819ms    remaining: 436ms
137:    learn: 14.2384227       total: 822ms    remaining: 429ms
138:    learn: 14.2383129       total: 824ms    remaining: 421ms
139:    learn: 14.2381284       total: 827ms    remaining: 414ms
140:    learn: 14.2379333       total: 831ms    remaining: 407ms
141:    learn: 14.2377842       total: 841ms    remaining: 403ms
142:    learn: 14.2377339       total: 848ms    remaining: 397ms
143:    learn: 14.2374857       total: 856ms    remaining: 392ms
144:    learn: 14.2374115       total: 864ms    remaining: 387ms
145:    learn: 14.2372437       total: 873ms    remaining: 383ms
146:    learn: 14.2371414       total: 883ms    remaining: 378ms
147:    learn: 14.2368901       total: 894ms    remaining: 375ms
148:    learn: 14.2366976       total: 898ms    remaining: 368ms
149:    learn: 14.2365054       total: 901ms    remaining: 361ms
150:    learn: 14.2364867       total: 904ms    remaining: 353ms
151:    learn: 14.2362995       total: 907ms    remaining: 346ms
152:    learn: 14.2362570       total: 909ms    remaining: 339ms
153:    learn: 14.2361636       total: 912ms    remaining: 332ms
154:    learn: 14.2360008       total: 916ms    remaining: 325ms
155:    learn: 14.2359841       total: 917ms    remaining: 318ms
156:    learn: 14.2358507       total: 921ms    remaining: 311ms
157:    learn: 14.2357303       total: 924ms    remaining: 304ms
158:    learn: 14.2356278       total: 927ms    remaining: 297ms
159:    learn: 14.2355114       total: 930ms    remaining: 291ms
160:    learn: 14.2353736       total: 934ms    remaining: 284ms
161:    learn: 14.2352139       total: 939ms    remaining: 278ms
162:    learn: 14.2351954       total: 942ms    remaining: 272ms
163:    learn: 14.2351590       total: 944ms    remaining: 265ms
164:    learn: 14.2350872       total: 947ms    remaining: 258ms
165:    learn: 14.2349845       total: 950ms    remaining: 252ms
166:    learn: 14.2348642       total: 959ms    remaining: 247ms
167:    learn: 14.2348100       total: 970ms    remaining: 242ms
168:    learn: 14.2347670       total: 976ms    remaining: 237ms
169:    learn: 14.2346680       total: 984ms    remaining: 231ms
170:    learn: 14.2346136       total: 989ms    remaining: 226ms
171:    learn: 14.2345644       total: 996ms    remaining: 220ms
172:    learn: 14.2345442       total: 1s       remaining: 214ms
173:    learn: 14.2345378       total: 1.01s    remaining: 208ms
174:    learn: 14.2344850       total: 1.01s    remaining: 203ms
175:    learn: 14.2344675       total: 1.02s    remaining: 197ms
176:    learn: 14.2344103       total: 1.02s    remaining: 190ms
177:    learn: 14.2343823       total: 1.02s    remaining: 184ms
```

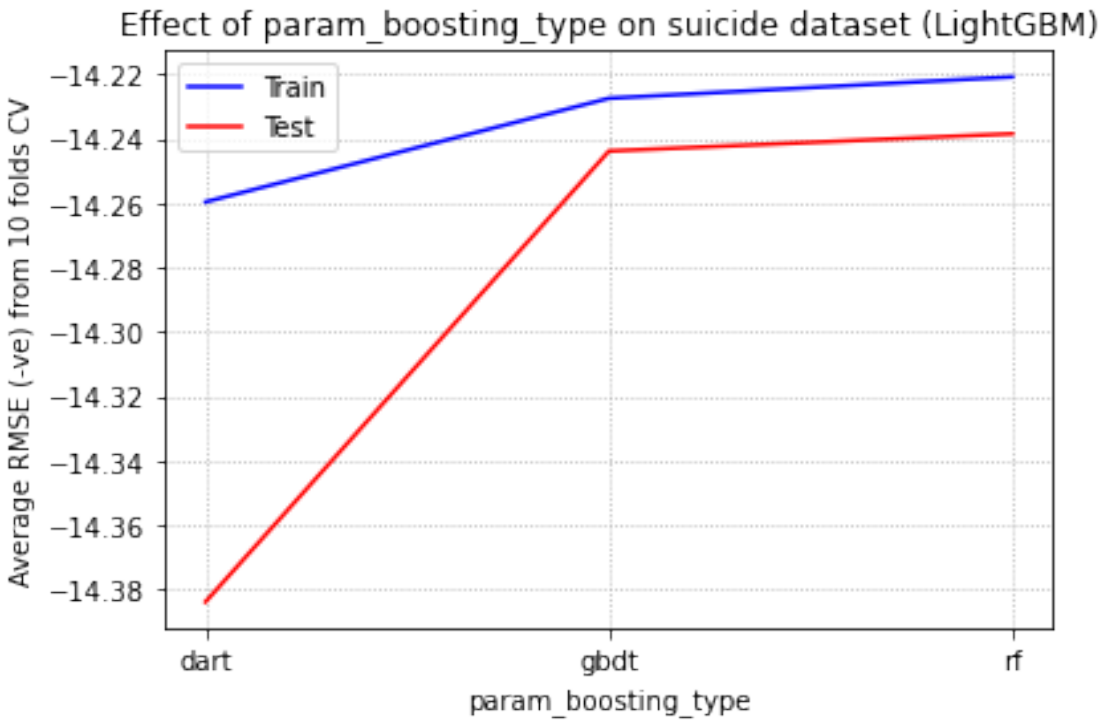
```
178:    learn: 14.2343456       total: 1.03s    remaining: 178ms
179:    learn: 14.2343382       total: 1.03s    remaining: 172ms
180:    learn: 14.2343268       total: 1.04s    remaining: 166ms
181:    learn: 14.2343186       total: 1.04s    remaining: 160ms
182:    learn: 14.2342674       total: 1.05s    remaining: 155ms
183:    learn: 14.2342410       total: 1.06s    remaining: 150ms
184:    learn: 14.2342046       total: 1.06s    remaining: 144ms
185:    learn: 14.2341857       total: 1.07s    remaining: 138ms
186:    learn: 14.2341097       total: 1.07s    remaining: 132ms
187:    learn: 14.2340749       total: 1.08s    remaining: 126ms
188:    learn: 14.2340485       total: 1.08s    remaining: 120ms
189:    learn: 14.2340289       total: 1.08s    remaining: 114ms
190:    learn: 14.2340142       total: 1.08s    remaining: 108ms
191:    learn: 14.2339531       total: 1.09s    remaining: 102ms
192:    learn: 14.2339447       total: 1.09s    remaining: 96ms
193:    learn: 14.2338954       total: 1.09s    remaining: 90.2ms
194:    learn: 14.2338787       total: 1.1s     remaining: 84.3ms
195:    learn: 14.2338675       total: 1.1s     remaining: 78.5ms
196:    learn: 14.2338426       total: 1.1s     remaining: 72.7ms
197:    learn: 14.2338232       total: 1.1s     remaining: 67ms
198:    learn: 14.2337598       total: 1.11s    remaining: 61.3ms
199:    learn: 14.2337380       total: 1.11s    remaining: 55.6ms
200:    learn: 14.2337185       total: 1.11s    remaining: 50ms
201:    learn: 14.2336727       total: 1.12s    remaining: 44.3ms
202:    learn: 14.2336433       total: 1.12s    remaining: 38.7ms
203:    learn: 14.2336203       total: 1.13s    remaining: 33.1ms
204:    learn: 14.2336039       total: 1.13s    remaining: 27.7ms
205:    learn: 14.2335888       total: 1.14s    remaining: 22.1ms
206:    learn: 14.2335793       total: 1.15s    remaining: 16.6ms
207:    learn: 14.2335776       total: 1.15s    remaining: 11.1ms
208:    learn: 14.2335687       total: 1.15s    remaining: 5.52ms
209:    learn: 14.2335574       total: 1.16s    remaining: 0us
```

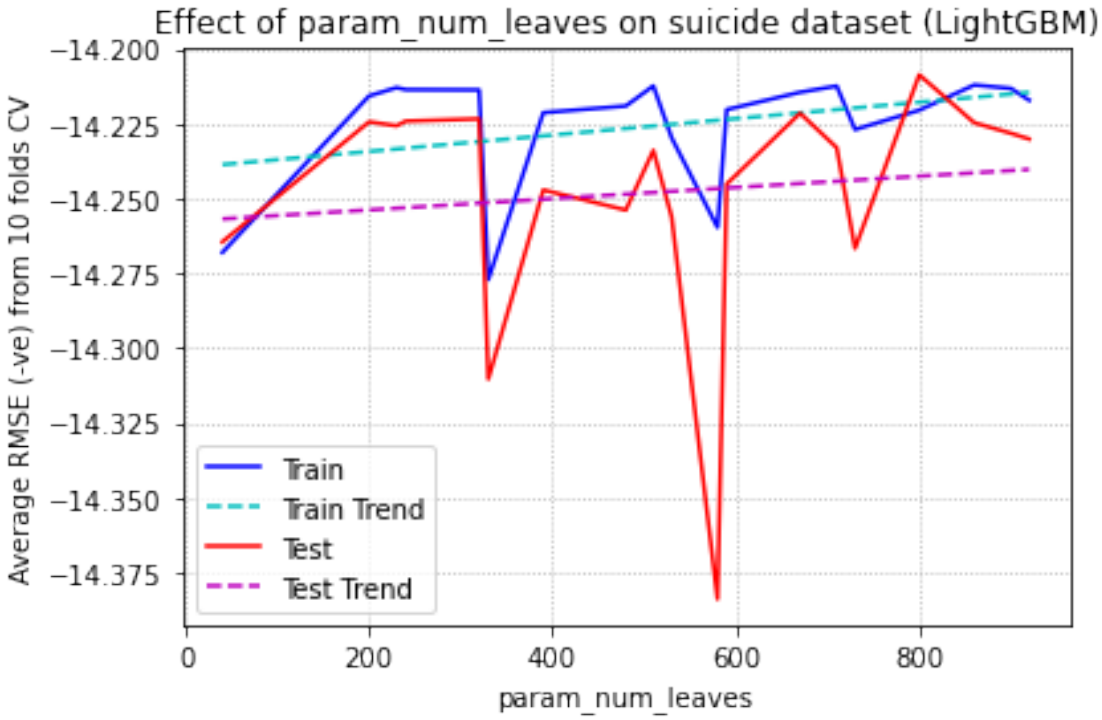
[373]:
```
print('Best parameters (suicide):',optcat.best_params_,',Test RMSE:',optcat.
 ↪best_score_)
print('Train RMSE:',min(optcat.cv_results_['mean_train_score']))
```

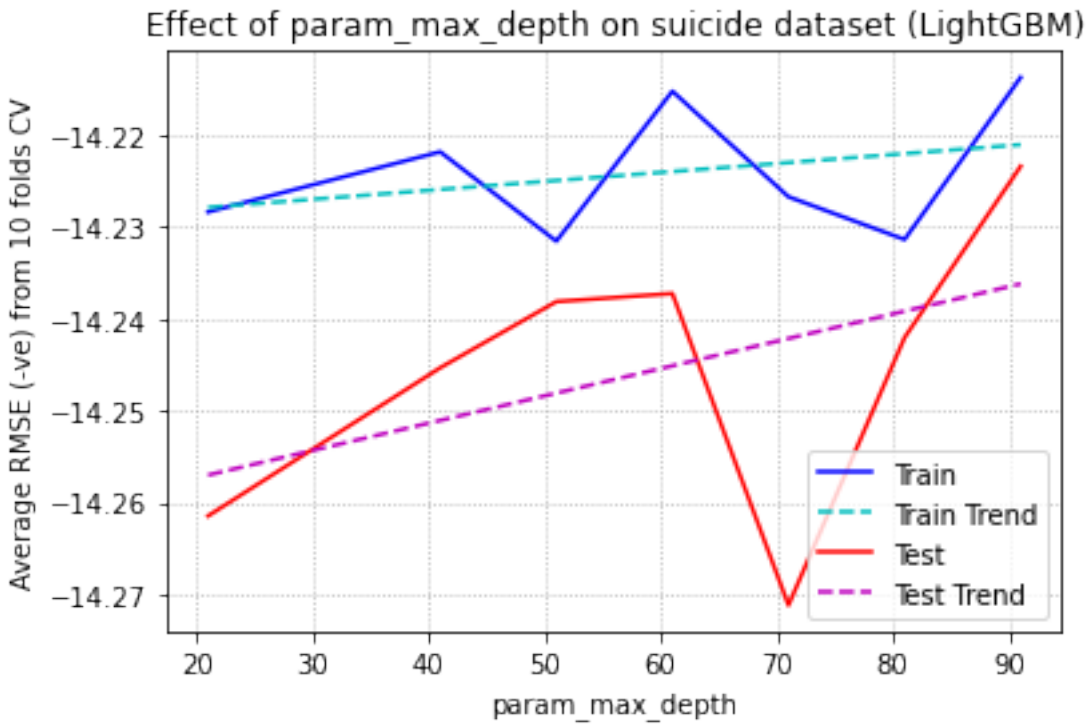
```
Best parameters (suicide): OrderedDict([('bagging_temperature', 0.1),
('colsample_bylevel', 0.8), ('grow_policy', 'SymmetricTree'), ('l2_leaf_reg',
0.01), ('max_depth', 9), ('num_trees', 210), ('score_function', 'Cosine')])
,Test RMSE: -14.220428456433268
Train RMSE: -17.23940884883195
```

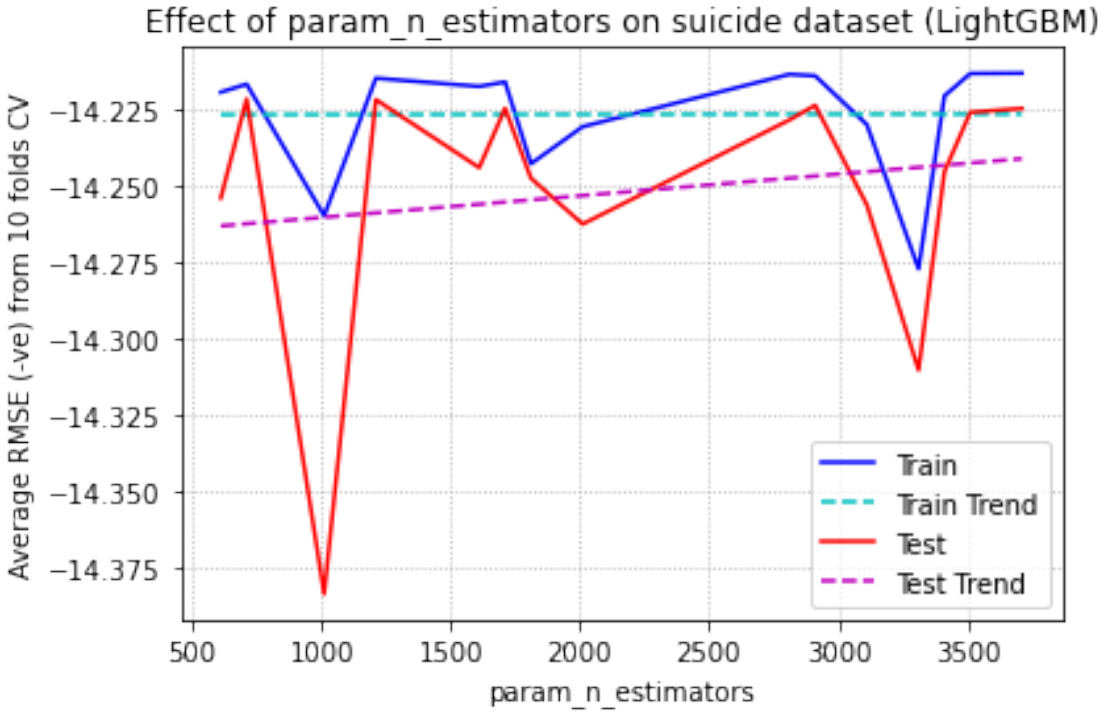
[490]:
```
param_list =␣
 ↪['param_boosting_type','param_num_leaves','param_max_depth','param_n_estimators',
            ␣
 ↪'param_reg_alpha','param_reg_lambda','param_subsample','param_subsample_freq',
```

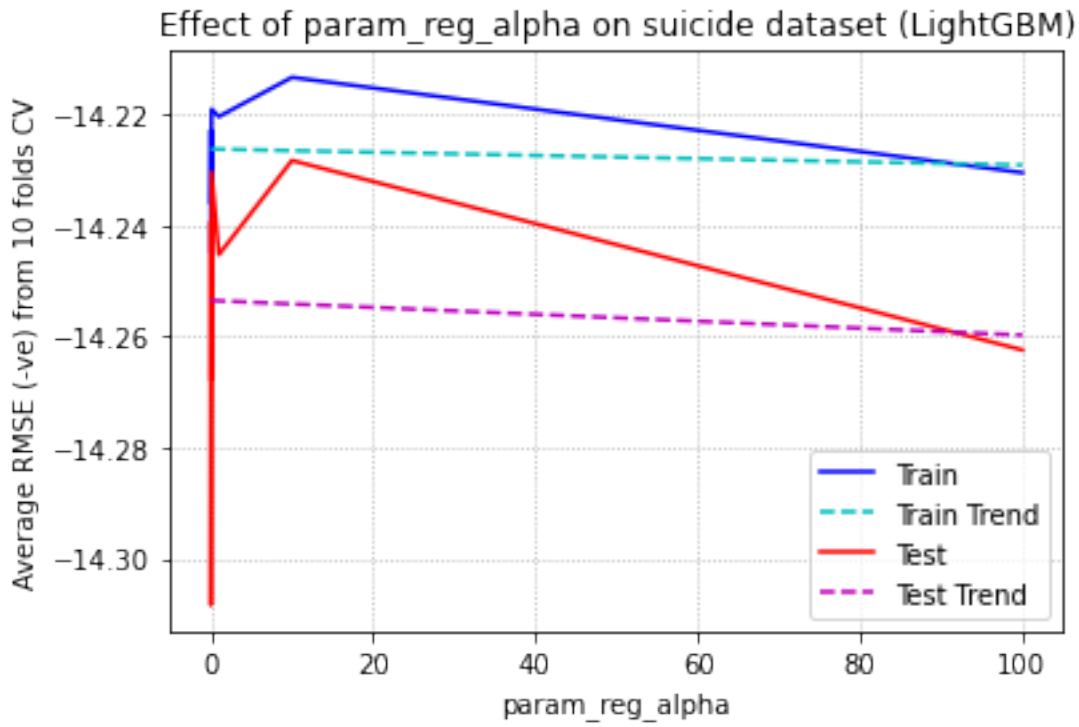
```
              'param_min_split_gain']
for param in param_list:
    param_set = sorted(list(set(opt.cv_results_[param])))
    param_trainscore = []
    param_testscore = []
    for item in param_set:
        param_trainscore.append(np.mean([opt.cv_results_['mean_train_score'][k]
                                    for k in [i for i, x in enumerate(opt.
 ↪cv_results_[param])
                                                if x == item]]))
        param_testscore.append(np.mean([opt.cv_results_['mean_test_score'][k]
                                    for k in [i for i, x in enumerate(opt.
 ↪cv_results_[param])
                                                if x == item]]))
    plt.plot(param_set,param_trainscore,label="Train",color='b')
    if(type(param_set[0]).__name__ != 'str'):
        plt.plot(param_set,np.poly1d(np.
 ↪polyfit(param_set,param_trainscore,1))(param_set),'--',label="Train␣
 ↪Trend",color='c')
    plt.plot(param_set,param_testscore,label="Test",color='r')
    if(type(param_set[0]).__name__ != 'str'):
        plt.plot(param_set,np.poly1d(np.
 ↪polyfit(param_set,param_testscore,1))(param_set),'--',label="Test␣
 ↪Trend",color='m')
    plt.legend()
    plt.grid(linestyle=':')
    plt.xlabel(param)
    plt.ylabel('Average RMSE (-ve) from 10 folds CV')
    plt.title("Effect of %s on suicide dataset (LightGBM)" % param)
    plt.savefig('Q26lightgbm'+param+'.png',dpi=300,bbox_inches='tight')
    plt.show()
```

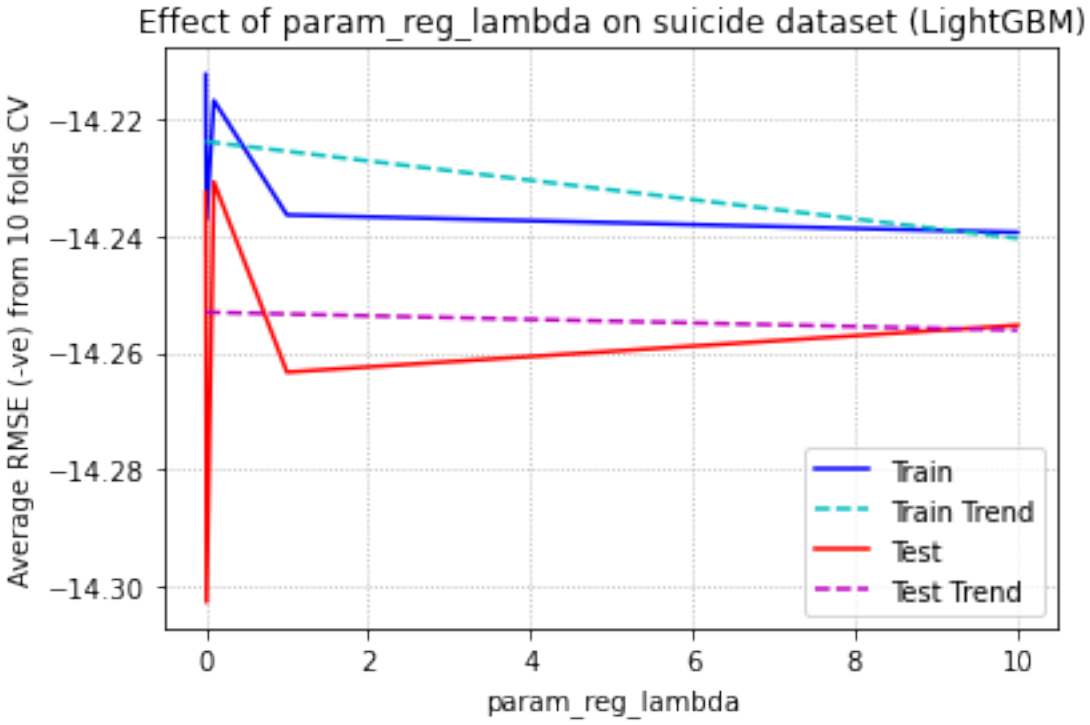

Effect of param_boosting_type on suicide dataset (LightGBM)
Train
Test
Average RMSE (-ve) from 10 folds CV
−14.22
−14.24
−14.26
−14.28
−14.30
−14.32
−14.34
−14.36
−14.38
dart
gbdt
rf
param_boosting_type

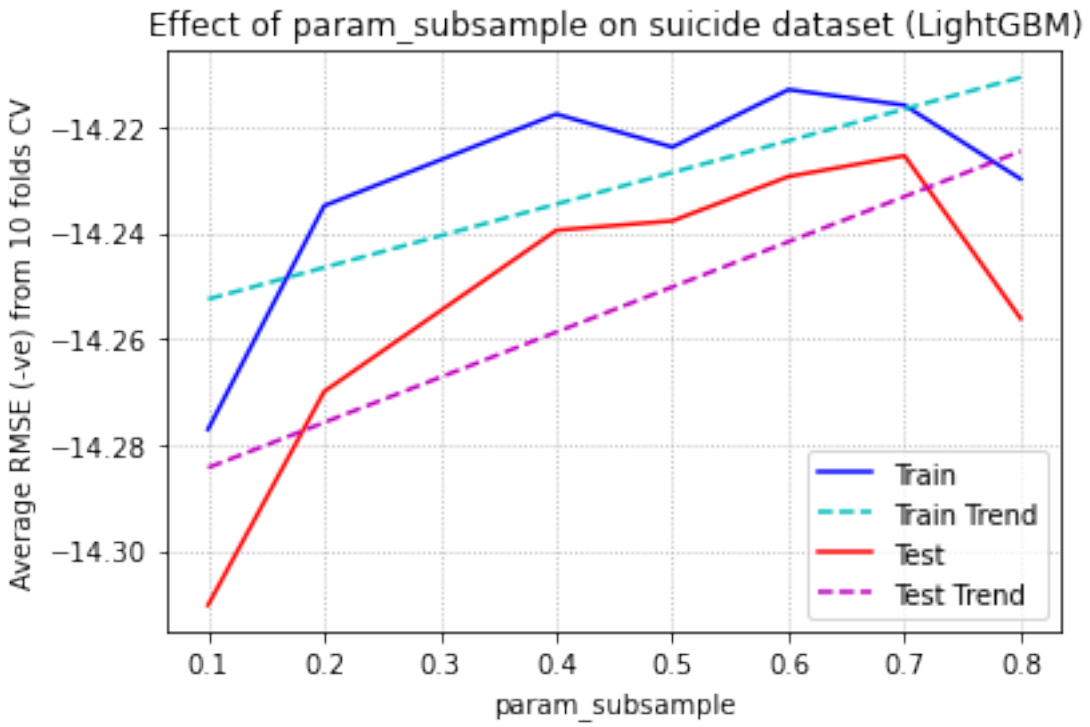

Effect of param_num_leaves on suicide dataset (LightGBM)
Average RMSE (-ve) from 10 folds CV
−14.200
−14.225
−14.250
−14.275
−14.300
−14.325
−14.350
−14.375
Train
Train Trend
Test
Test Trend
0
200
400
600
800
param_num_leaves

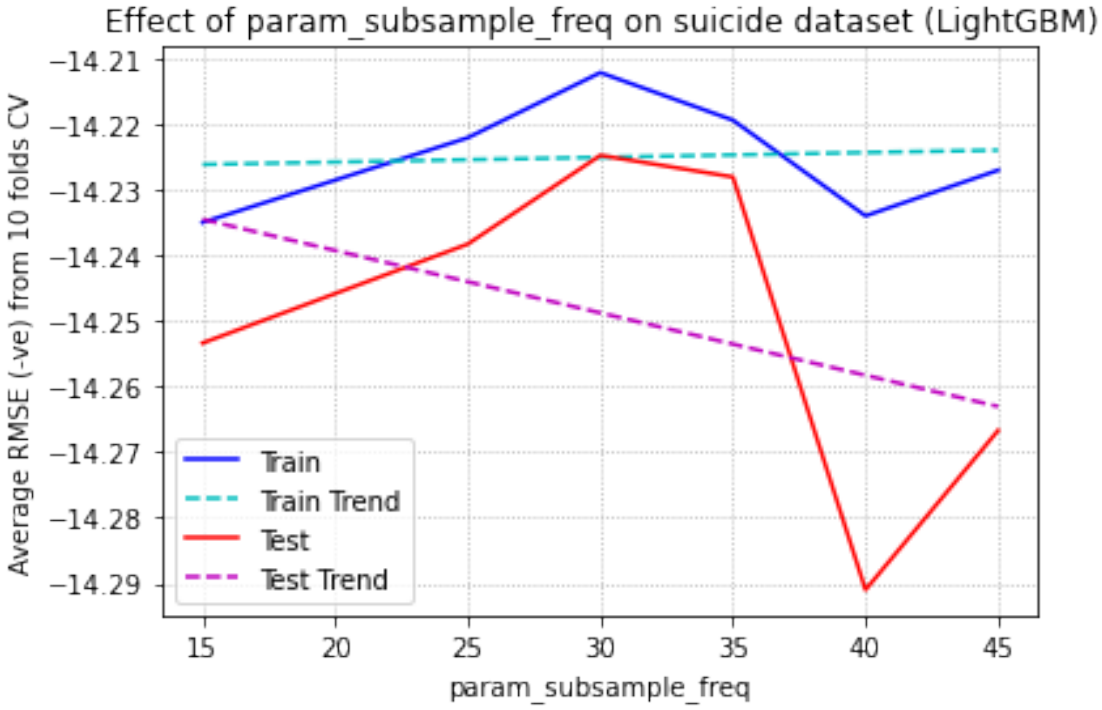

Effect of param_max_depth on suicide dataset (LightGBM)
Average RMSE (-ve) from 10 folds CV
−14.22
−14.23
−14.24
−14.25
−14.26
−14.27
20
30
40
50
60
70
80
90
param_max_depth
Train
Train Trend
Test
Test Trend

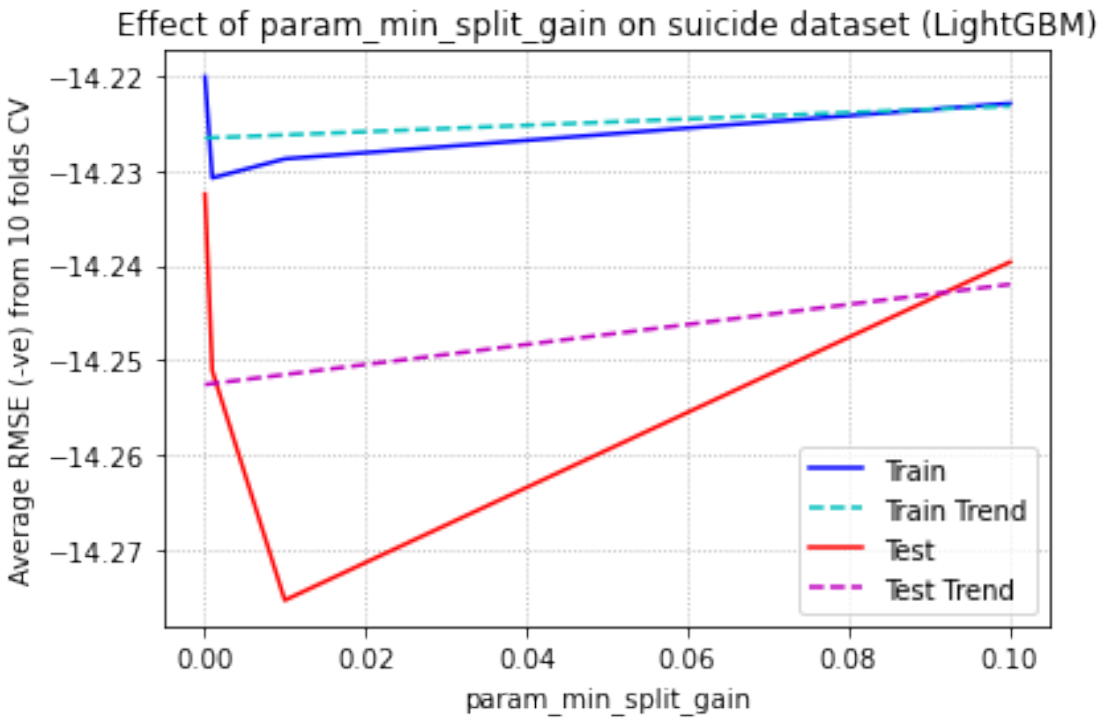

Effect of param_n_estimators on suicide dataset (LightGBM)
Average RMSE (-ve) from 10 folds CV
−14.225
−14.250
−14.275
−14.300
−14.325
−14.350
−14.375
500
1000
1500
2000
2500
3000
3500
param_n_estimators
Train
Train Trend
Test
Test Trend


Effect of param_reg_alpha on suicide dataset (LightGBM)
Average RMSE (-ve) from 10 folds CV
−14.22
−14.24
−14.26
−14.28
−14.30
0
20
40
60
80
100
param_reg_alpha
Train
Train Trend
Test
Test Trend


Effect of param_reg_lambda on suicide dataset (LightGBM)
Average RMSE (-ve) from 10 folds CV
−14.22
−14.24
−14.26
−14.28
−14.30
0
2
4
6
8
10
param_reg_lambda
Train
Train Trend
Test
Test Trend


Effect of param_subsample on suicide dataset (LightGBM)
Average RMSE (-ve) from 10 folds CV
−14.22
−14.24
−14.26
−14.28
−14.30
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
param_subsample
Train
Train Trend
Test
Test Trend


Effect of param_subsample_freq on suicide dataset (LightGBM)
Average RMSE (-ve) from 10 folds CV
−14.21
−14.22
−14.23
−14.24
−14.25
−14.26
−14.27
−14.28
−14.29
15
20
25
30
35
40
45
param_subsample_freq
Train
Train Trend
Test
Test Trend

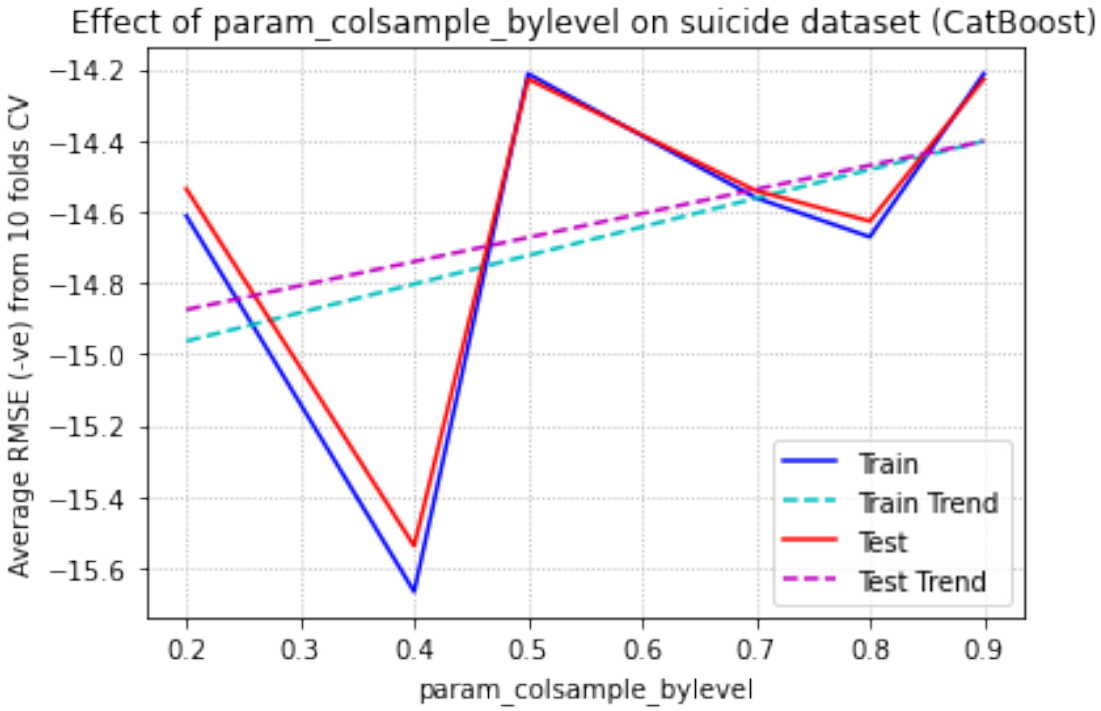


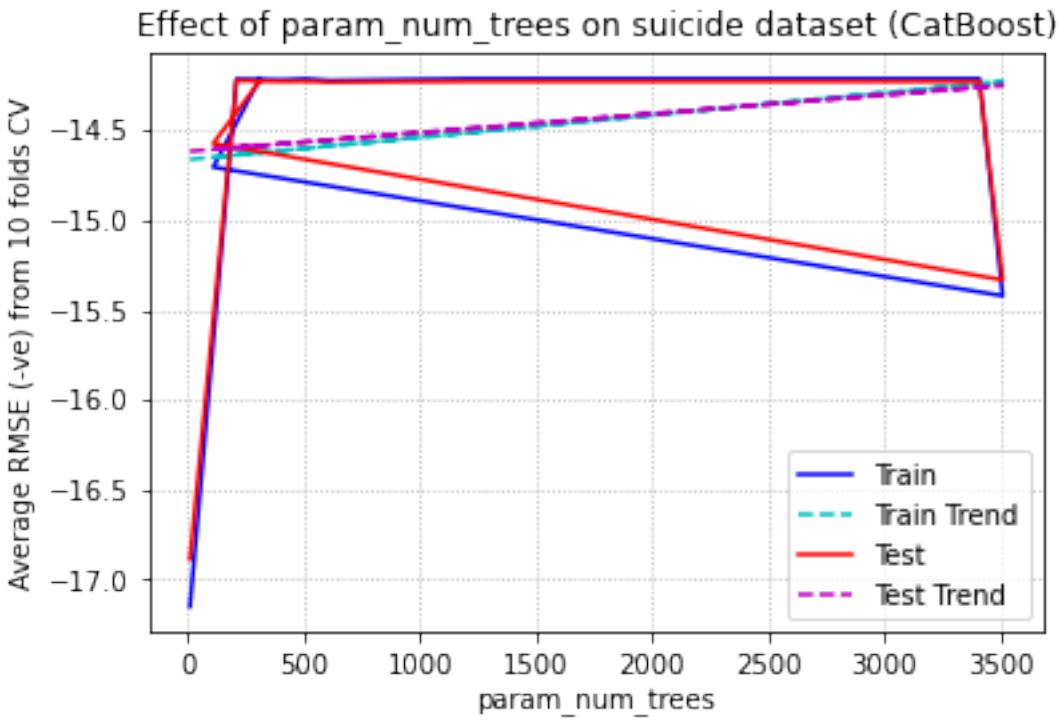
[607]:
```
def join_list(param_set_1,param_set_2,param_score_1,param_score_2):
    names = param_set_1 + param_set_2
    results_values = param_score_1 + param_score_2
    averages = {}
    counts = {}
    for name, value in zip(names, results_values):
        if name in averages:
            averages[name] += value
            counts[name] += 1
        else:
            averages[name] = value
            counts[name] = 1
    for name in averages:
        averages[name] = averages[name]/float(counts[name])
    comb_param_set = list(averages.keys())
    comb_score = list(averages.values())
    return comb_param_set, comb_score
```

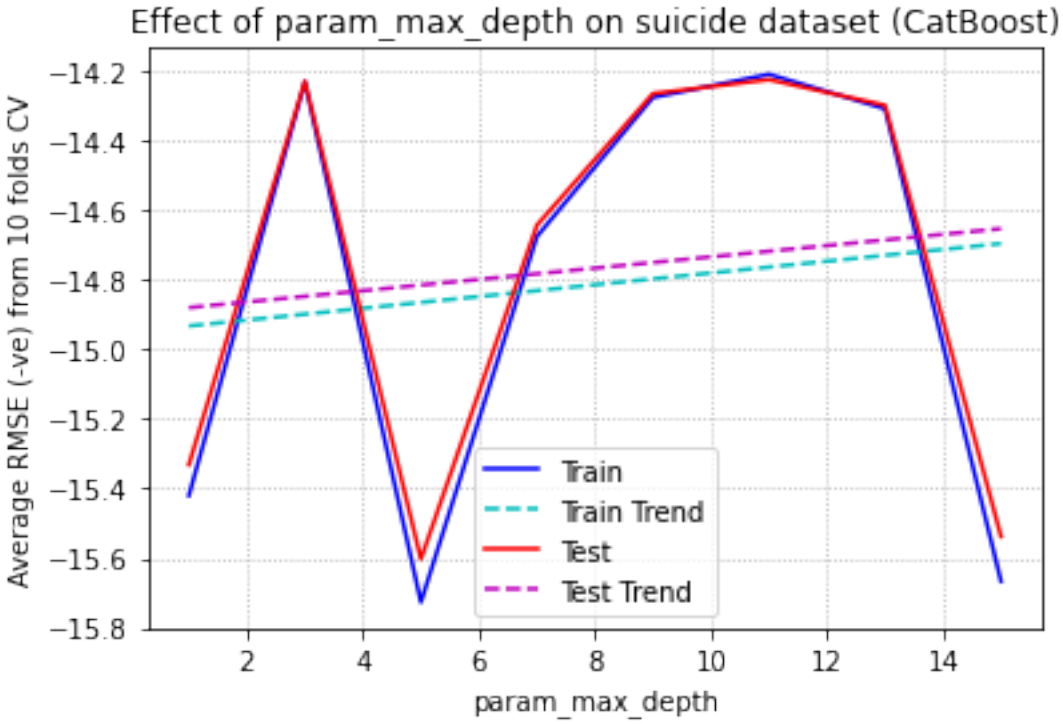
[609]:
```
param_list =␣
 ↪['param_colsample_bylevel','param_num_trees','param_max_depth','param_l2_leaf_reg',
              'param_num_leaves','param_bagging_temperature']
for param in param_list:
    param_set = sorted(list(set(optcatLG.cv_results_[param])))
```

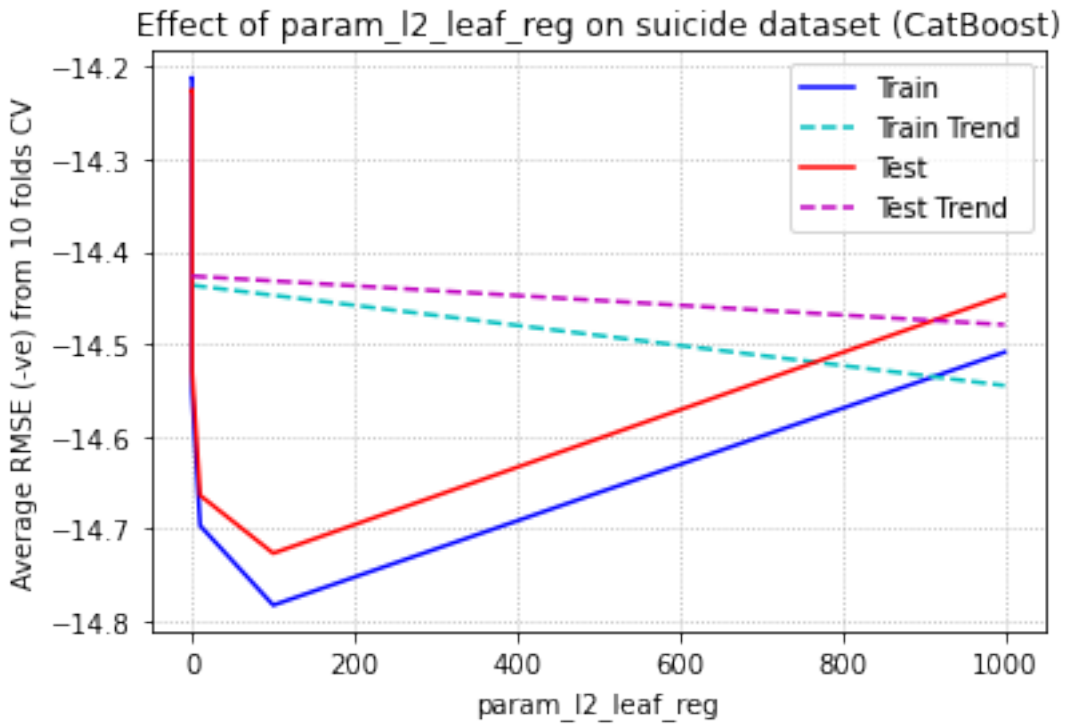
```
    param_set_2 = sorted(list(set(optcat.cv_results_[param])))
    param_trainscore = []
    param_trainscore_2 = []
    param_testscore = []
    param_testscore_2 = []
    for item in param_set:
        param_trainscore.append(np.mean([optcatLG.
↪cv_results_['mean_train_score'][k]
                                     for k in [i for i, x in enumerate(optcatLG.
↪cv_results_[param])
                                               if x == item]]))
        param_testscore.append(np.mean([optcatLG.
↪cv_results_['mean_test_score'][k]
                                     for k in [i for i, x in enumerate(optcatLG.
↪cv_results_[param])
                                               if x == item]]))
    for it in param_set_2:
        param_trainscore_2.append(np.mean([optcat.
↪cv_results_['mean_train_score'][k]
                                     for k in [i for i, x in enumerate(optcat.
↪cv_results_[param])
                                               if x == it]]))
        param_testscore_2.append(np.mean([optcat.
↪cv_results_['mean_test_score'][k]
                                     for k in [i for i, x in enumerate(optcat.
↪cv_results_[param])
                                               if x == it]]))

    comb_param_set, comb_trainscore =␣
↪join_list(param_set,param_set_2,param_trainscore,param_trainscore_2)
    comb_param_set, comb_testscore =␣
↪join_list(param_set,param_set_2,param_testscore,param_testscore_2)

    plt.plot(comb_param_set,comb_trainscore,label="Train",color='b')
    if(type(comb_param_set[0]).__name__ != 'str'):
        plt.plot(comb_param_set,np.poly1d(np.
↪polyfit(comb_param_set,comb_trainscore,1))(comb_param_set),'--',label="Train␣
↪Trend",color='c')
    plt.plot(comb_param_set,comb_testscore,label="Test",color='r')
    if(type(comb_param_set[0]).__name__ != 'str'):
        plt.plot(comb_param_set,np.poly1d(np.
↪polyfit(comb_param_set,comb_testscore,1))(comb_param_set),'--',label="Test␣
↪Trend",color='m')
    plt.legend()
    plt.grid(linestyle=':')
    plt.xlabel(param)
```

```
    plt.ylabel('Average RMSE (-ve) from 10 folds CV')
    plt.title("Effect of %s on suicide dataset (CatBoost)" % param)
    plt.savefig('Q26catboost'+param+'.png',dpi=300,bbox_inches='tight')
    plt.show()
```

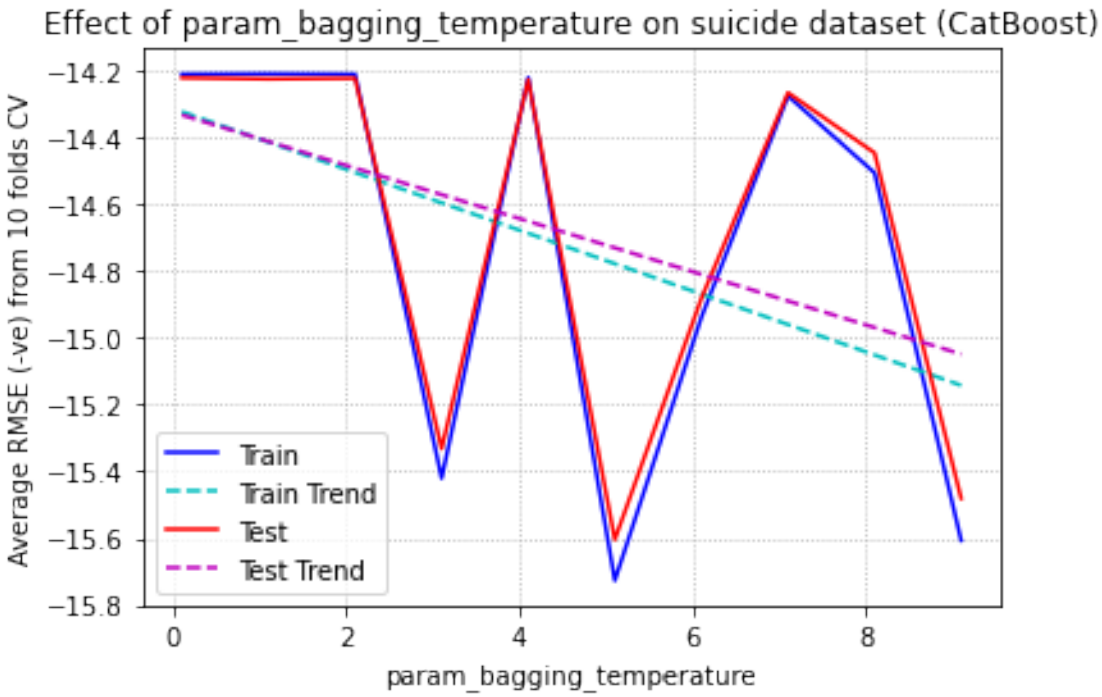



Effect of param_num_trees on suicide dataset (CatBoost)
Average RMSE (-ve) from 10 folds CV
−14.5
−15.0
−15.5
−16.0
−16.5
−17.0
0
500
1000
1500
2000
2500
3000
3500
param_num_trees
Train
Train Trend
Test
Test Trend


Effect of param_max_depth on suicide dataset (CatBoost)
Average RMSE (-ve) from 10 folds CV
−14.2
−14.4
−14.6
−14.8
−15.0
−15.2
−15.4
−15.6
−15.8
2
4
6
8
10
12
14
param_max_depth
Train
Train Trend
Test
Test Trend

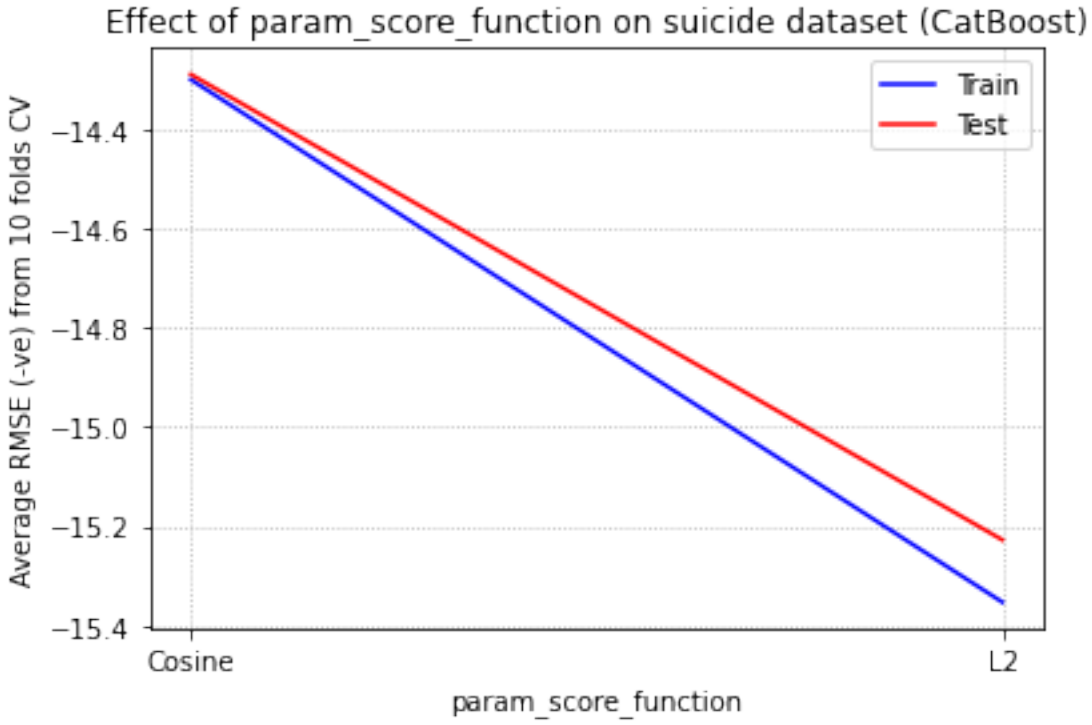

Effect of param_l2_leaf_reg on suicide dataset (CatBoost)
Average RMSE (-ve) from 10 folds CV
param_l2_leaf_reg
Train
Train Trend
Test
Test Trend
−14.2
−14.3
−14.4
−14.5
−14.6
−14.7
−14.8
0
200
400
600
800
1000


Effect of param_num_leaves on suicide dataset (CatBoost)
Average RMSE (-ve) from 10 folds CV
param_num_leaves
Train
Train Trend
Test
Test Trend
−14.5
−15.0
−15.5
−16.0
−16.5
−17.0
200
400
600
800



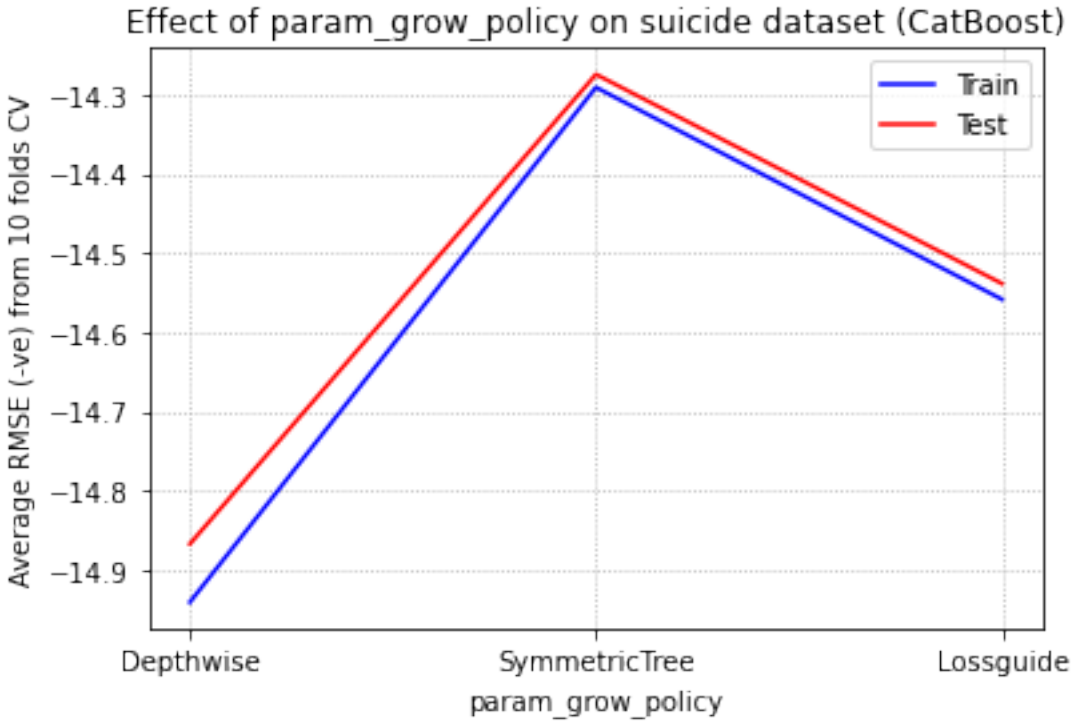
[611]:
```
param_list = ['param_score_function']
for param in param_list:
    param_set = sorted(list(set(optcat.cv_results_[param])))
    param_trainscore = []
    param_testscore = []
    for item in param_set:
        param_trainscore.append(np.mean([optcat.
↪cv_results_['mean_train_score'][k]
                                         for k in [i for i, x in enumerate(optcat.
↪cv_results_[param])
                                                   if x == item]]))
        param_testscore.append(np.mean([optcat.cv_results_['mean_test_score'][k]
                                         for k in [i for i, x in enumerate(optcat.
↪cv_results_[param])
                                                   if x == item]]))
    plt.plot(param_set,param_trainscore,label="Train",color='b')
    if(type(param_set[0]).__name__ != 'str'):
        plt.plot(param_set,np.poly1d(np.
↪polyfit(param_set,param_trainscore,1))(param_set),'--',label="Train␣
↪Trend",color='c')
    plt.plot(param_set,param_testscore,label="Test",color='r')
    if(type(param_set[0]).__name__ != 'str'):
```

```
        plt.plot(param_set,np.poly1d(np.
↪polyfit(param_set,param_testscore,1))(param_set),'--',label="Test␣
↪Trend",color='m')
    plt.legend()
    plt.grid(linestyle=':')
    plt.xlabel(param)
    plt.ylabel('Average RMSE (-ve) from 10 folds CV')
    plt.title("Effect of %s on suicide dataset (CatBoost)" % param)
    plt.savefig('Q26catboost'+param+'.png',dpi=300,bbox_inches='tight')
    plt.show()
```



[619]:
```
param_list = ['param_grow_policy']
for param in param_list:
    param_set = sorted(list(set(optcat.cv_results_[param])))
    param_trainscore = []
    param_testscore = []
    for item in param_set:
        param_trainscore.append(np.mean([optcat.
↪cv_results_['mean_train_score'][k]
                                    for k in [i for i, x in enumerate(optcat.
↪cv_results_[param])
                                              if x == item]]))
        param_testscore.append(np.mean([optcat.cv_results_['mean_test_score'][k]
```

```
                                        for k in [i for i, x in enumerate(optcat.
↪cv_results_[param])
                                                  if x == item]]))
    param_trainscore.append(np.mean(optcatLG.cv_results_['mean_train_score']))
    param_testscore.append(np.mean(optcatLG.cv_results_['mean_test_score']))
    param_set.append('Lossguide')
    plt.plot(param_set,param_trainscore,label="Train",color='b')
    if(type(param_set[0]).__name__ != 'str'):
        plt.plot(param_set,np.poly1d(np.
↪polyfit(param_set,param_trainscore,1))(param_set),'--',label="Train␣
↪Trend",color='c')
    plt.plot(param_set,param_testscore,label="Test",color='r')
    if(type(param_set[0]).__name__ != 'str'):
        plt.plot(param_set,np.poly1d(np.
↪polyfit(param_set,param_testscore,1))(param_set),'--',label="Test␣
↪Trend",color='m')
    plt.legend()
    plt.grid(linestyle=':')
    plt.xlabel(param)
    plt.ylabel('Average RMSE (-ve) from 10 folds CV')
    plt.title("Effect of %s on suicide dataset (CatBoost)" % param)
    plt.savefig('Q26catboost'+param+'.png',dpi=300,bbox_inches='tight')
    plt.show()
```



[ ]: