Importing libraries:

The necessary libraries, functions and methods were imported.

Reading the data:

Using the read_excel function from Pandas, the data was read into a dataframe object 'df' in the notebook.

```
In [4]: df.head()
Out[4]:
```

	custid	region	townsize	gender	age	agecat	birthmonth	ed	edcat	jobcat	...	owncd	ownpda	ownpc
0	3964-QJWTRG-NPN	1	2.0	1	20	2	September	15	3	1	...	0	0	0
1	0648-AIPJSP-UVM	5	5.0	0	22	2	May	17	4	2	...	1	1	1
2	5195-TLUDJE-HVO	3	4.0	1	67	6	June	14	2	2	...	1	0	0
3	4459-VLPQUH-3OL	4	3.0	0	23	2	May	16	3	2	...	1	0	1

Understanding the data:

```
In [10]: df.shape
Out[10]: (5000, 130)

In [11]: df.columns
Out[11]: Index(['custid', 'region', 'townsize', 'gender', 'age', 'agecat', 'birthmonth',
                'ed', 'edcat', 'jobcat',
                ...
                'owncd', 'ownpda', 'ownpc', 'ownipod', 'owngame', 'ownfax', 'news',
                'response_01', 'response_02', 'response_03'],
               dtype='object', length=130)

In [12]: df.info()
         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 5000 entries, 0 to 4999
         Columns: 130 entries, custid to response_03
```

The data has 5,000 instances and 130 features. There are 31 float features, 97 integer features, and 2 object features.

In [13]:
```
df.describe()
```

Out[13]:

	region	townsize	gender	age	agecat	ed	edcat
count	5000.00000	4998.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000
mean	3.00140	2.687275	0.503600	47.025600	4.238800	14.543000	2.672000
std	1.42176	1.425925	0.500037	17.770338	1.308785	3.281083	1.211738
min	1.00000	1.000000	0.000000	18.000000	2.000000	6.000000	1.000000
25%	2.00000	1.000000	0.000000	31.000000	3.000000	12.000000	2.000000
50%	3.00000	3.000000	1.000000	47.000000	4.000000	14.000000	2.000000
75%	4.00000	4.000000	1.000000	62.000000	5.000000	17.000000	4.000000
max	5.00000	5.000000	1.000000	79.000000	6.000000	23.000000	5.000000

8 rows × 128 columns

In [14]:
```
df.dtypes
```

Out[14]:
```
custid          object
region           int64
townsize       float64
gender           int64
```

The describe function only gives a summary of numerical data. Therefore, 128 out of 130 features have numerical values. Two features- ‘custid’ and ‘birthmonth’ are of type object. ‘cardspent’ and ‘card2spent’ are the features providing the credit card spend for the primary and secondary card for a particular customer. The aim is to predict the total card spend, that is, cardspent+card2spent

Preparing the data:

The data has to be cleaned and prepared in order to make it fit for further analysis. The data was checked to know if there are null values present or not.

```
In [15]: df.isnull().sum().value_counts()

Out[15]: 0       115
         2         3
         3296      2
         3656      2
         2622      2
         3         2
         1         2
         1422      1
         1419      1
         dtype: int64
```

The data has 115 features with zero null values, 3 features with 2 null values, 2 features with 3296 null values. 2 features with 3656 null values, 2 features with 2622 null values, 2 features with 3 null values, 2 features with 1 null value, 1 feature with 1422 null values, and 1 feature with 1419 null values.

```
In [16]: df.isnull().sum()

Out[16]: custid          0
         region          0
         townsize        2
         gender          0
         age             0
         agecat          0
         birthmonth      0
         ed              0
         edcat           0
         jobcat          0
         union           0
         employ          0
         empcat          0
```

Then the count of null values was checked feature-wise. The features with more than 20% of their data being null values were dropped entirely. Further null values were treated after splitting the dataset into its respective numerical and categorical parts.

Using the data dictionary, the dataset 'df' was split into a categorical dataset – 'cat' and a numerical dataset –'num'.

```
In [23]: cat.shape
Out[23]: (5000, 80)

In [24]: num.shape
Out[24]: (5000, 47)
```

The categorical dataset has 80 features, and the numerical dataset has 47 features. The null values in each dataset are treated differently. First, the null values in the categorical data are replaced with the mode of the respective data. The null values in the numerical data are replaced with the mean of the respective data.

In the numerical dataset, a features 'totalspend' was added, with values equal to cardspent+card2spent. This will be the target or the dependent variable for the analysis. Moreover, the outliers in the numerical data are clipped with a lower threshold for 0.01 percentile and an upper threshold of 0.99 percentile.

```
num.columns

Index(['custid', 'age', 'ed', 'income', 'lninc', 'debtinc', 'creddebt',
       'lncreddebt', 'othdebt', 'lnothdebt', 'spoused', 'reside', 'pets',
       'pets_cats', 'pets_dogs', 'pets_birds', 'pets_reptiles', 'pets_small',
       'pets_saltfish', 'pets_freshfish', 'carvalue', 'commutetime',
       'carditems', 'card2items', 'tenure', 'longmon', 'lnlongmon', 'longten',
       'lnlongten', 'tollmon', 'tollten', 'equipmon', 'equipten', 'cardmon',
       'cardten', 'wiremon', 'wireten', 'hourstv', 'employ', 'cardspent',
       'card2spent', 'address', 'cars', 'cardtenure', 'card2tenure',
       'totalspendln'],
      dtype='object')
```

Some features in the numerical dataset provide repetitive information, for example, 'income' and 'lninc' provide income and log of the income. Such features are dropped.

The categorical data is converted to dummified data, using the get_dummies function in Pandas. The original features are dropped, and the categorical dataset now consists of dummified features of the earlier categorical information. This dummified data is saved as the ‘dumm’ dataset.

```
dumm.shape
```

```
(5000, 172)
```

The treated numerical and categorical dataset are merged, based on the common features ‘custid’, and saved to the dataset ‘df’. The features ‘custid’ is dropped.

```
df=pd.merge(num,dumm,on='custid')
```

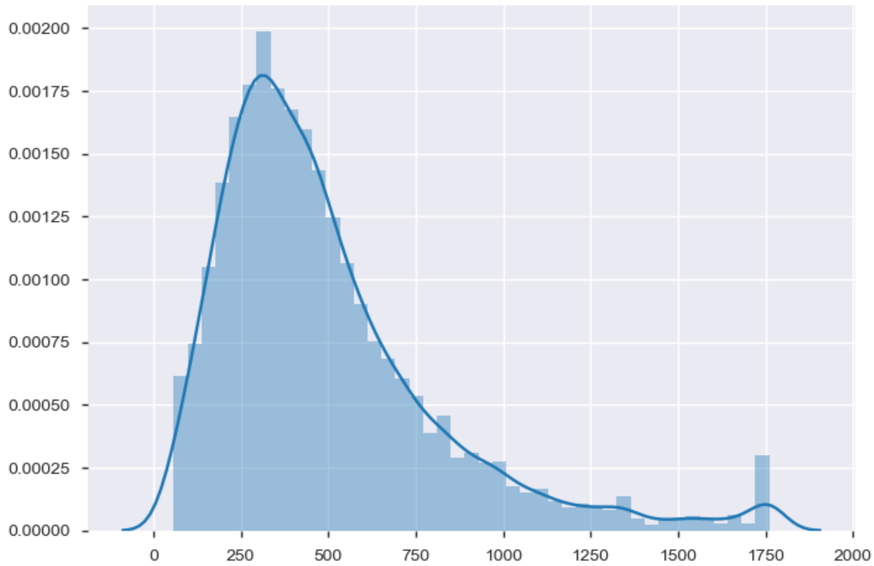
```
df.drop('custid',axis=1,inplace=True)
```

```
df.shape
```

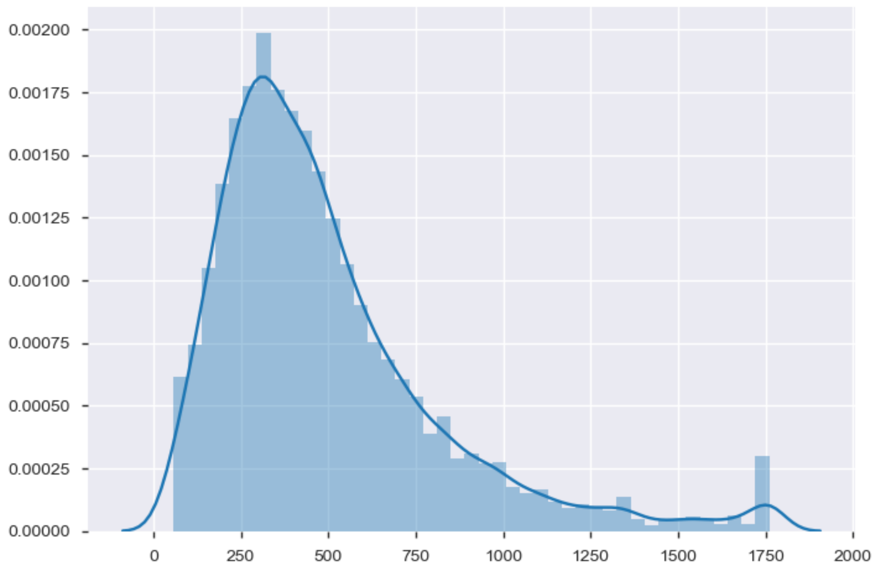
```
(5000, 217)
```

The target variable should preferably have a normal distribution for better fit into a model. The target variable ‘totalspend’ is checked for its distribution.

<matplotlib.axes._subplots.AxesSubplot at 0x2af54c78208>

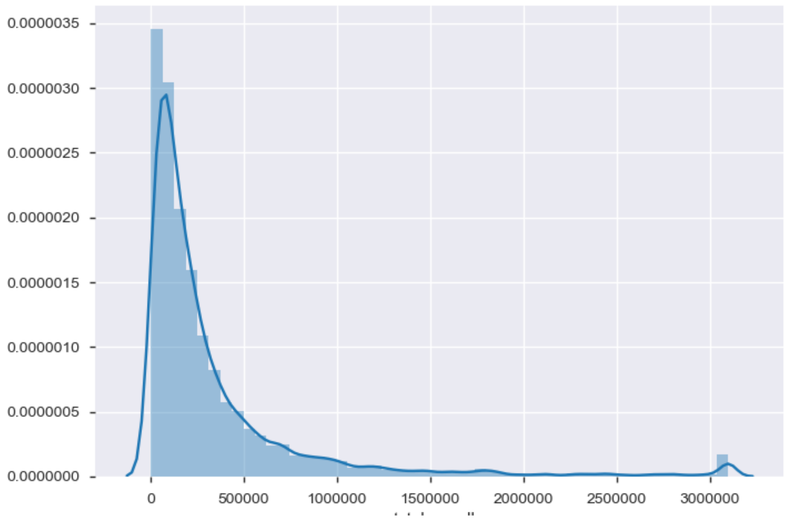


<matplotlib.axes._subplots.AxesSubplot at 0x2af54c78208>



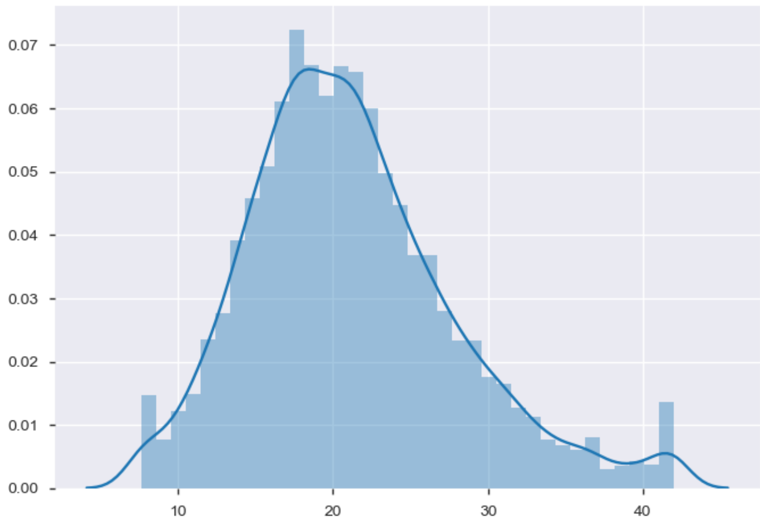
Since the feature ‘totalspend’ does not have a perfect normal distribution, some operations can be performed on the features to try and get it closer to normal distribution spread. The operations tried on the features were squaring it, taking the cube, taking the square root, taking the cube root, taking the log, etc. The spread for the square of totalspend was:

<matplotlib.axes._subplots.AxesSubplot at 0x2af54961898>

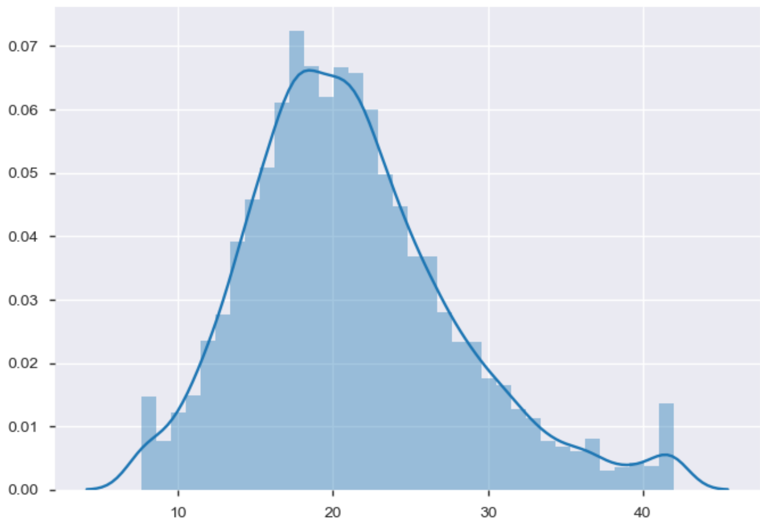


It is evident that the square of totalspend was skewed, and not normally distributed. The spread for square root of totalspend was:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2af54120ac8>
```

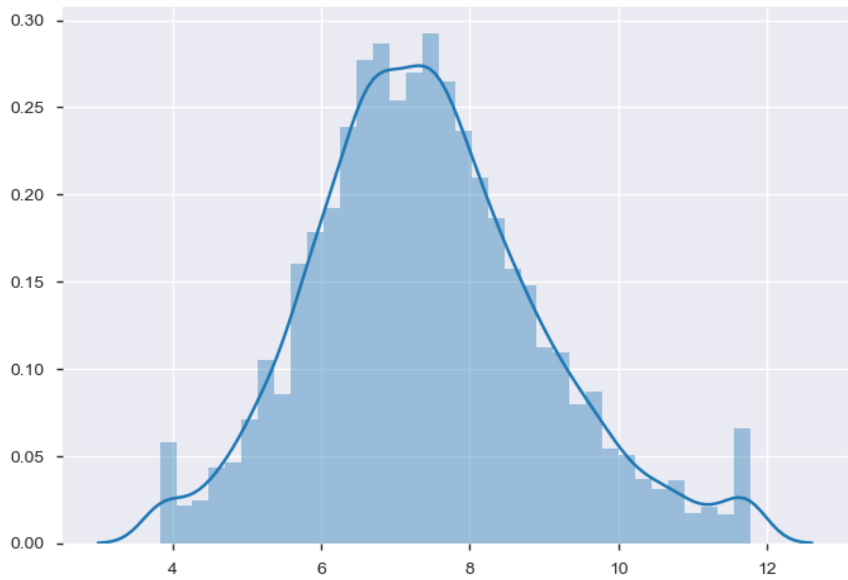



```
<matplotlib.axes._subplots.AxesSubplot at 0x2af54120ac8>
```



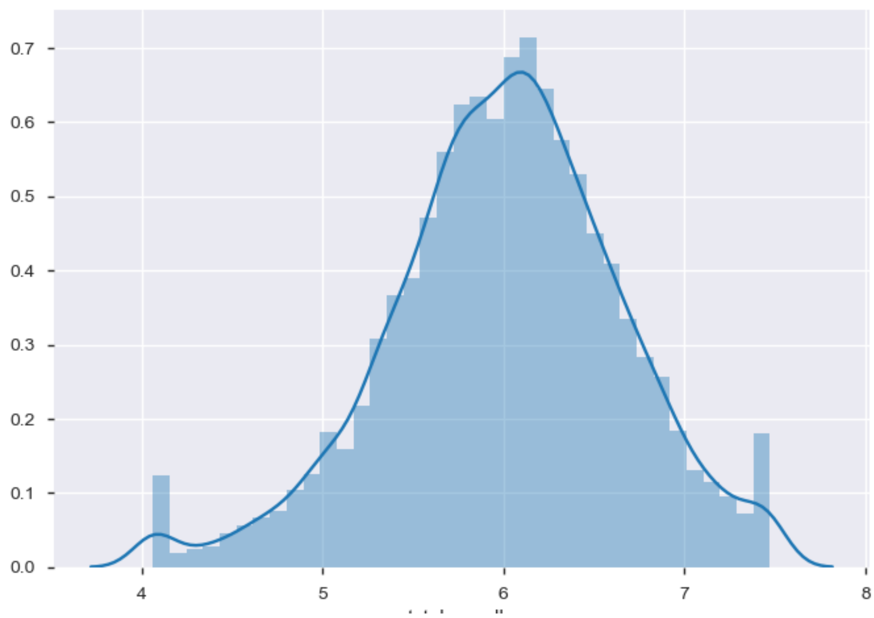
It is evident that the square root of the variable is also skewed. The spread for cube root of totalspend was:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2af5494a4e0>
```



The spread for cube root of the variable totalspend was considerably better, but still skewed. The spread for log of totalspend was:

```
: <matplotlib.axes._subplots.AxesSubplot at 0x2af540e14a8>
```



The closest operation on totalspend to a normal distribution was the log of totalspend. Therefore, totalspendln was added as a features to the dataset and totalspend was dropped.

Feature selection:

Using pandas profiling, the features with high inter-correlation were recognised and dropped.

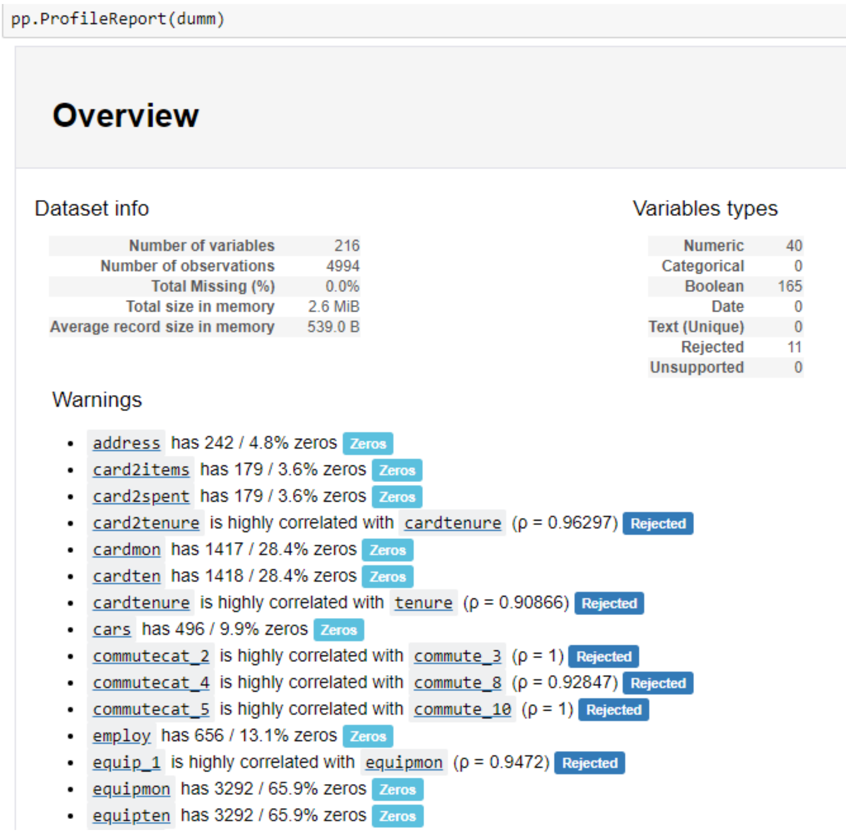


Figure 1: Credit card spend dataset Pandas Profile report

To the features left after Pandas profiling, the dataset is checked for correlation using the corr() function in Pandas. The correlation table is saved to excel and the excel file is analysed to select the features correlated with 'totalspend'. The correlation range considered is between 0.1 to 0.7 and -0.7 to -0.1. These are the features which will be used for further analysis.

The features selected after checking for correlation are treated to VIF. Variance Inflation Factor is applied to detect any further multicollinearity in the selected features. To implement this, variance_inflation_factor is imported from statsmodels.stats.outliers_influence. The features, after being treated to VIF are saved to the dataset 'dfvif'.

The features, after being treated to VIF, are subjected to F-regression. The F-test for linear regression tests whether any of the independent variables in a multiple linear regression model are significant. It tests each variable for its individual effect on the dependent variable. It is essentially a feature scoring procedure. F-regression is implemented by importing SelectKBest, f_regression from sklearn.feature_selection.

Model Building and Evaluation:

After f-regression, the features selected to fit into a model are:

cardspent, card2spent, othdebt, carcatvalue_3, creddebt, inccat_5, reitre_1, inccat_4, reason_2, card_2, card_3, inccat_3, agecat_5, wireten, carcatvalue_2, card2_3, tollten, equipten, agecat_4, gender_1, card2_2, jobcat_2, card_4, ownpda1, hometype_2, ownfax_1, response_03_1, card4, inccat_2, pager_1, and card2_4.

Since the target variable 'totalspendln' is a continuous variable and not a Boolean variable, linear regression is chosen over logistic regression. Logistic regression is suitable in classification models, where the target variable is a Boolean value.

A multiple linear regression model refers to multiple variables to make a prediction. Let 'x1','x2','x3' etc. be the predictors, or independent variables, and let 'y' be the dependent or target variable. Then, y can be represented as:

$$y= ax1+bx2+cx3+m$$

where m is the intercept and a, b, and c are the coefficients/slopes for x1, x2 , and x3 respectively.

To fit the dataset with the selected features into a model, the dataset is split into training and testing parts. This is done using the train_test_split from sklearn.model_selection. The training dataset is then fit into a linear regression model 'lm' and based on the trained model, predictions are made for the testing dataset. This is done using LinearRegression from sklearn.linear_model.

```
score=lm.score(test_X, test_y)
print(score*100)
```

```
88.62977955524775
```

```
score=lm.score(train_X,train_y)
print(score*100)
```

```
89.01035949291361
```

The model gives us an 89.01% prediction score in the training dataset, and 88.62% prediction score in the testing dataset. We use OLS stats models to obtain a summary of the model and the coefficients for each variable.

OLS Regression Results

Dep. Variable:	totalspendln	**R-squared:**	0.888
Model:	OLS	**Adj. R-squared:**	0.887
Method:	Least Squares	**F-statistic:**	1226.
Date:	Tue, 28 May 2019	**Prob (F-statistic):**	0.00
Time:	15:36:49	**Log-Likelihood:**	563.47
No. Observations:	4994	**AIC:**	-1061.
Df Residuals:	4961	**BIC:**	-845.9
Df Model:	32		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	4.7797	0.018	271.839	0.000	4.745	4.814
cardspent	0.0016	2.56e-05	61.010	0.000	0.002	0.002
card2spent	0.0010	4.77e-05	21.507	0.000	0.001	0.001
othdebt	0.0011	0.001	0.983	0.326	-0.001	0.003
carcatvalue_3	0.0097	0.015	0.633	0.527	-0.020	0.040
creddebt	0.0002	0.002	0.138	0.890	-0.003	0.004
inccat_5	0.1405	0.023	6.149	0.000	0.096	0.185

The training dataset was further fit into Decision Tree, Random Forest, Lasso Regression, and Ridge Regression models.

Decision trees are simple but intuitive models that utilize a top-down approach in which the root node creates binary splits until a certain criteria is met.

Random forests are a popular ensemble method that can be used to build predictive models for both classification and regression problems.

LASSO stands for *Least Absolute Shrinkage and Selection Operator*. Lasso regression performs **L1 regularization**, i.e. it adds a factor of sum of absolute value of coefficients in the optimization objective. Thus, lasso regression optimizes the following:

$$\text{Objective} = \text{RSS} + \alpha * (\text{sum of absolute value of coefficients})$$

Ridge regression performs '**L2 regularization**', i.e. it adds a factor of sum of squares of coefficients in the optimization objective. Thus, ridge regression optimizes the following:

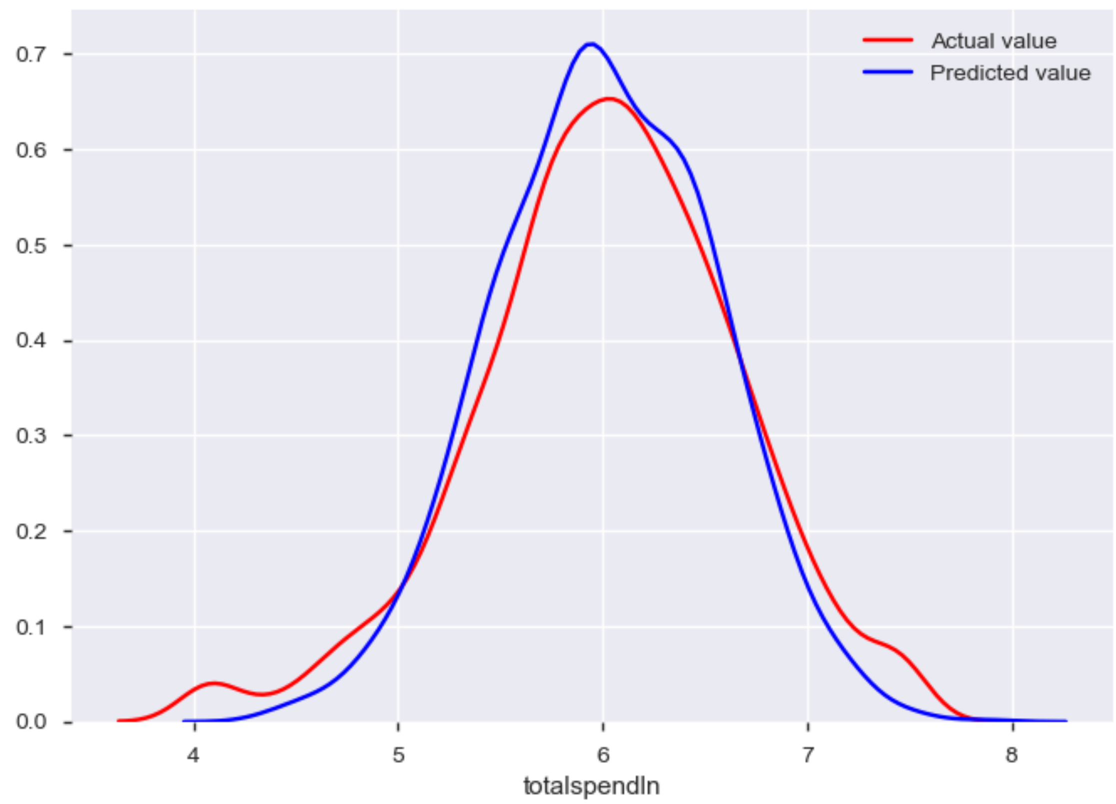
$$\text{Objective} = \text{RSS} + \alpha * (\text{sum of square of coefficients})$$

Here, α (alpha) is the parameter which balances the amount of emphasis given to minimizing RSS vs. minimizing sum of square of coefficients.

Model	Train score	Test score
Decision Tree	0.7846	0.7554
Random Forest	0.9331	0.7342
Lasso Regression	0.8571	0.8105
Ridge Regression	0.7989	0.7913
Linear Regression	0.8901	0.8862

The OLS stats summary shows that the adjusted R^2 score is 0.887. The score should be close to 1. It is therefore evident that the linear regression model is a good fit. Comparing the scores of all the models, the best train and test scores were achieved by applying the linear regression model. The random forest model provided good accuracy in the training dataset, but an average accuracy in the testing dataset, which suggests that it was over fitted. The 'dfmodel' field in the summary gives a count of the independent variables, that is, 32. These 32 features were the best fit out of the 217 features in the original dataset. The 'coef' features provides the coefficient for all the independent variables. The AIC and BIC fields are indicators for overfitting and underfitting. The lower the scores, the better. Since the AIC and BIC scores are negative, it indicates that the model is a good fit. The graph below depicts a comparison between the actual values of the test dataset and the predicted values:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2af567395f8>
```



Results Obtained:

The final linear regression model, when tested for actual values against predicted values, gave an accuracy of 88.62% for the test dataset, and 89.01% for the train dataset.

```
score=lm.score(test_X, test_y)
print(score*100)
```

```
88.62977955524775
```

```
score=lm.score(train_X,train_y)
print(score*100)
```

```
89.01035949291361
```

We use OLS stats models to obtain a summary of the model and the coefficients for each variable.

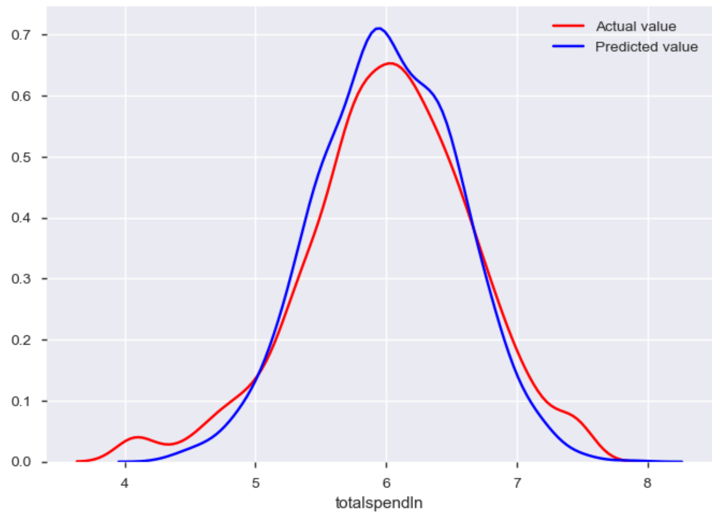
OLS Regression Results

Dep. Variable:	totalspendln	**R-squared:**	0.888
Model:	OLS	**Adj. R-squared:**	0.887
Method:	Least Squares	**F-statistic:**	1226.
Date:	Tue, 28 May 2019	**Prob (F-statistic):**	0.00
Time:	15:36:49	**Log-Likelihood:**	563.47
No. Observations:	4994	**AIC:**	-1061.
Df Residuals:	4961	**BIC:**	-845.9
Df Model:	32		
Covariance Type:	nonrobust		

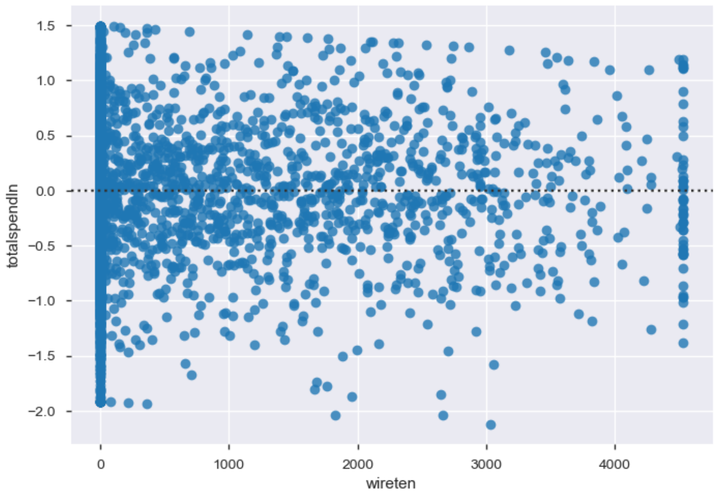
	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	4.7797	0.018	271.839	0.000	4.745	4.814
cardspent	0.0016	2.56e-05	61.010	0.000	0.002	0.002
card2spent	0.0010	4.77e-05	21.507	0.000	0.001	0.001
othdebt	0.0011	0.001	0.983	0.326	-0.001	0.003
carcatvalue_3	0.0097	0.015	0.633	0.527	-0.020	0.040
creddebt	0.0002	0.002	0.138	0.890	-0.003	0.004
inccat_5	0.1405	0.023	6.149	0.000	0.096	0.185

The OLS stats summary shows that the adjusted R^2 score is 0.887. The score should be close to 1. It is therefore evident that the linear regression model is a good fit. Comparing the scores of all the models, the best train and test scores were achieved by applying the linear regression model. The ‘dfmodel’ field in the summary gives a count of the independent variables, that is, 32. These 32 features were the best fit out of the 217 features in the original dataset. The ‘coef’ column provides the coefficient for all the independent variables. The AIC and BIC fields are indicators for overfitting and underfitting. The lower the scores, the better. Since the AIC and BIC scores are negative, it indicates that the model is a good fit. The graph below depicts a comparison between the actual values of the test dataset and the predicted values:

<matplotlib.axes._subplots.AxesSubplot at 0x2af567395f8>



The residual plot obtained for the dependent/target variable, when plotted against a correlated variable 'wireten' was:



This has no pattern or curvature. Therefore, the model is a good fit.