Introduction to Machine Learning - Part 3: Review on New Datasets

Introduction

This article is a continuation of the previous two articles [1,2] that gave a brief introduction to a number of different algorithms. This series of articles have been devoted to the general machine learning operations associated with supervised learning. This class of supervised learning algorithms are all dependent upon having a clean dataset to operate on. The first articles [3,4] on exploratory data analysis gave some insights into how to examine data, how to clean data, how to prepare data for algorithm consumption, and how to visualize it for improved understanding or improved explanation to the consumer.

The supervised learning algorithms used all needed a large subset of the dataset known as the training data and a smaller subset of the dataset know as the test set. The test set data are used to determine how well the model could predict unseen target variable values that are either continuous (regression) or discrete (classification).

The models built so far were using different algorithms for regression and classification problems. A number of the algorithms were shared because they have both regressor and classifier versions of the algorihtm that can be used on the specific type of problem. This article focuses on applying those algorithms to new datasets for similiar problems.

Earier algorithms on linear regression, logistic regression, KNN, decision trees and random forests (bagging verses boosting versions) will be applied to new datasets. Additionally, a new algorithm, Support Vector Machines (SVM), will be introduced. The new datasets include: Melbourne Housing values, Kickstarter project success, advertising click through, and Berlin Airbnb pricing.

Finally, the results obtained with the algorithms are evaluated for measuring model performance. This is done using mean square error (MSE), confusion matrix and classification reports, depending on if it's a regression or classification problem. Lastly, optimizations are considered with some model's hyperparameters. This can be done manually or with techniques like grid search or cross-validation (k-folds).

This article closely follows the exposition contained in [5]. All of the datasets used here have URLs listed in [4]. The reader is encouraged to review the source material and download the datasets for additional study. Repeating and practicing the data cleaning techniques used here on the supplied datasets will help build familiarity with options available for new datasets. Applying the algorithms to these datasets will provide a more in-depth education of how to build models for prediction.

Theory

This section covers the details of the algorithms being demonstrated. The details of the bulk of the algorithms can be found in the previous articles [1,2] and are not repeated here. The one new algorithm being introduced in this article is Support Vector Machines (SVM). Details about its definition, purpose, how it works and performs, what kind of datasets it is best suited to, and any of its hyperparameters that can be used for optimization will be discussed here.

Support Vector Machines (SVM)

The section introduces the SVM [6]. It is another supervised learning algorithm designed for regression. It can be used for prediction of both continuous (numerical regression) and discrete (classification) target variables. It is very good for classification problems because it better handles outliers. It can be used in binary or multi-category classification.

Logistic regression is usually the first algorithm that is thought of for classification problems. For binary cases, it uses the sigmoid function to separate data points into two classes. It attempts to draw a line that represents the boundary between the two classes of data. Then for new data points, based on their features, it attempts to classify the new data point into one of the two classes based on where it falls on either side of the line. Hence the prediction of which class it belongs to.

SVM differs from logistic regression because it uses what is known as a 'margin'. Unlike in logistic regression, where a single line is determined as the separator of two classes, here a wide strip is used to separate data classes. [5] provides one definition of margin as: "Its key feature is the margin, which is the distance beteween the boundary line and the nearest data point, multiplied by two."

Pictorially this image from [7] shows margin between the two classes.  Additionally, some very good figures of generalized hyperplanes separating two classes can be found in [8].

The margin basically provides a robust buffer between data points in the two classes that are separated by a boundary line. This buffer acts as a more stringent check against outlier data points that might appear much closer to one category's set of data points than the other. This added robustness, by having a strip of separation between the two sets of data points, will aide classification predictions of new data points. The wider buffer area makes for a more solid distinction between the classes, new data points that might have outlier characteristics crossing over to the other side are less likely to be mis-classified into the wrong class.

Reference [7] covers some high level details about the applicability of SVM to different kinds of datasets. SVM in general is not well suited to larger datasets. Large datasets can slow down training, so SVM is best applied to smaller datasets. If the data is too noisy, SVM does not perform as well. Lastly, hyperparameter tuning can be difficult. Adjusting the parameters for an optimal solution can take a long cycle of trail and error, or extra compute to grid search through the combination of parameters.

Applying ML Algorithms to New Datasets

Readers of the previous articles in the series will recognize that the Analysis and Experimental Results sections were treated separately. This article is a review of the algorithms, so those section headings will be expounded for each algorithm. Under each section will be subheadings for the Analysis and Experimental Results associated with dataset cleaning and model training to evaluate prediction performance.

Brief explanations of each dataset and its cleaning will be covered first. This will help reduce the quantity of data to be handled in model training. Then the algorithm will be selected and applied to produce a model. Additionally, optimization steps will be considered as part of improving the pedictions.

In [10]:
```python
# Commonly imported libraries used by the following examples.
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

Linear Regression

Supervised learning uses linear regression to train models for predicting continuous numeric target variables. This process attempts to fit a line to a known set of data points. It tries to minimize the distance between the data points and the line.

Analysis

This example uses the dataset of Melbourne (Australia) housing to train a linear regression model. It contains a large number of independent variables and many examples. The data is not perfect and needs to be cleaned before it can be used. Examining the data and culling the unnecessary columns is the first step to preparing it for training.

In [15]:
```
melbourne_housing_df = pd.read_csv('Melbourne_housing_FULL.csv')
melbourne_housing_df.head()
```

Out[15]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	...	Bathroom	Car	La
0	Abbotsford	68 Studley St	2	h	NaN	SS	Jellis	3/09/2016	2.5	3067.0	...	1.0	1.0	
1	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5	3067.0	...	1.0	1.0	
2	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5	3067.0	...	1.0	0.0	
3	Abbotsford	18/659 Victoria St	3	u	NaN	VB	Rounds	4/02/2016	2.5	3067.0	...	2.0	1.0	
4	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5	3067.0	...	2.0	0.0	

5 rows × 21 columns

In [16]:
```
melbourne_housing_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34857 entries, 0 to 34856
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Suburb         34857 non-null  object 
 1   Address        34857 non-null  object 
 2   Rooms          34857 non-null  int64  
 3   Type           34857 non-null  object 
 4   Price          27247 non-null  float64
 5   Method         34857 non-null  object 
 6   SellerG        34857 non-null  object 
 7   Date           34857 non-null  object 
 8   Distance       34856 non-null  float64
 9   Postcode       34856 non-null  float64
 10  Bedroom2       26640 non-null  float64
 11  Bathroom       26631 non-null  float64
 12  Car            26129 non-null  float64
 13  Landsize       23047 non-null  float64
 14  BuildingArea   13742 non-null  float64
 15  YearBuilt      15551 non-null  float64
 16  CouncilArea    34854 non-null  object 
 17  Lattitude      26881 non-null  float64
 18  Longtitude     26881 non-null  float64
 19  Regionname     34854 non-null  object 
 20  Propertycount  34854 non-null  float64
dtypes: float64(12), int64(1), object(8)
memory usage: 5.6+ MB
```

In [17]:
```
melbourne_housing_df.describe()
```

Out[17]:

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize
count	34857.000000	2.724700e+04	34856.000000	34856.000000	26640.000000	26631.000000	26129.000000	23047.000000
mean	3.031012	1.050173e+06	11.184929	3116.062859	3.084647	1.624798	1.728845	593.598993
std	0.969933	6.414671e+05	6.788892	109.023903	0.980690	0.724212	1.010771	3398.841946
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000	0.000000
25%	2.000000	6.350000e+05	6.400000	3051.000000	2.000000	1.000000	1.000000	224.000000
50%	3.000000	8.700000e+05	10.300000	3103.000000	3.000000	2.000000	2.000000	521.000000
75%	4.000000	1.295000e+06	14.000000	3156.000000	4.000000	2.000000	2.000000	670.000000
max	16.000000	1.120000e+07	48.100000	3978.000000	30.000000	12.000000	26.000000	433014.000000

From examining the dataset and its properties, it can be seen that many of the non-numeric data can be removed because they play no role in aiding the predictions (or they can not be easily converted to a usable numeric quantity). A number of them are also duplicate in description, so they do not add value by informing the house price that another variable is also doing. So those are eliminated and the contents examined again.

In [19]:

```
del melbourne_housing_df['Address']
del melbourne_housing_df['Method']
del melbourne_housing_df['SellerG']
del melbourne_housing_df['Date']
del melbourne_housing_df['Postcode']
del melbourne_housing_df['YearBuilt']
del melbourne_housing_df['Type']
del melbourne_housing_df['Lattitude']
del melbourne_housing_df['Longtitude']
del melbourne_housing_df['Regionname']
del melbourne_housing_df['Suburb']
del melbourne_housing_df['CouncilArea']
melbourne_housing_df.head()
```

Out[19]:

	Rooms	Price	Distance	Bedroom2	Bathroom	Car	Landsize	BuildingArea	Propertycount
0	2	NaN	2.5	2.0	1.0	1.0	126.0	NaN	4019.0
1	2	1480000.0	2.5	2.0	1.0	1.0	202.0	NaN	4019.0
2	2	1035000.0	2.5	2.0	1.0	0.0	156.0	79.0	4019.0
3	3	NaN	2.5	3.0	2.0	1.0	0.0	NaN	4019.0
4	3	1465000.0	2.5	3.0	2.0	0.0	134.0	150.0	4019.0

A number of columns have been removed from the dataset. Looking at the info() statistics earlier for the remaining columns, there are a number of rows that don't have values. The next set of operations to conduct on the data include managing the missing data. Different strategies can be employed for dealing with the missing information. It mainly depends on the type of data it is, what is missing, what would be a reasonable approximate value that could be filled in, what would be the consequence of using substitute values. Additional possibilities include removing the empty rows or removing the features (columns) from the dataset.

In all those cases, trade offs must be considered to evaluate the feasibility of the approaches used to clean the data. As the size of the dataset becomes smaller, it could lead to overfitting in the model training, or other side-effects that affect the prediction quality. It can also affect which algorithms can or can not be used because of the nature of the algorithm and its ability to process and learn from a dataset of that size.

In [21]:
```
# In this dataset, the quantity of null (missing) data needs to be identified first.
melbourne_housing_df.isnull().sum()
```

Out[21]:
```
Rooms                0
Price             7610
Distance             1
Bedroom2          8217
Bathroom          8226
Car               8728
Landsize         11810
BuildingArea     21115
Propertycount        3
dtype: int64
```

In [22]:
```
# Examining the shape can give a percentage estimate of how much data is in the dataset.
```

```
melbourne_housing_df.shape
```

Out[22]:

```
(34857, 9)
```

In [23]:

```
(melbourne_housing_df.isnull().sum() / melbourne_housing_df.shape[0])*100
```

Out[23]:

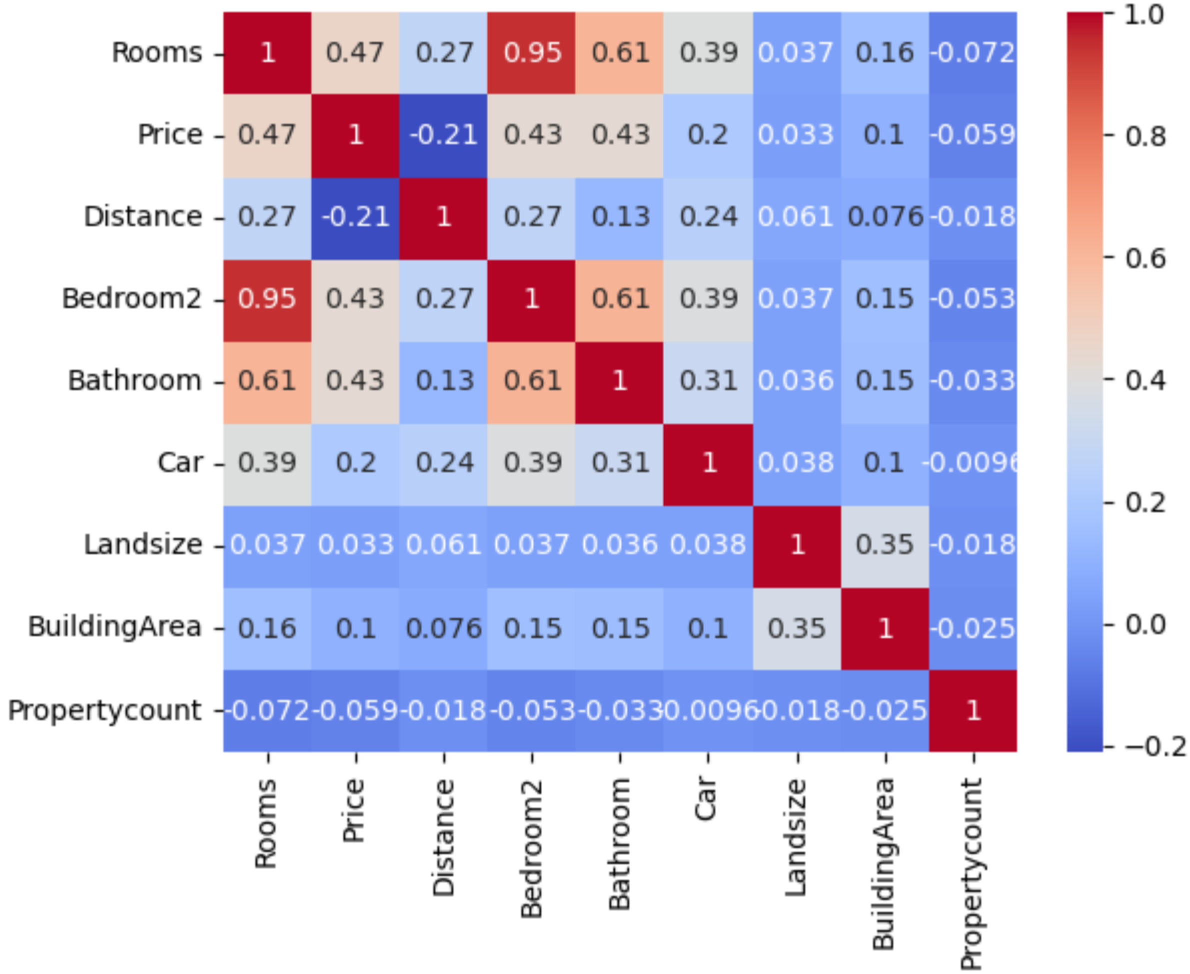
```
Rooms             0.000000
Price            21.832057
Distance          0.002869
Bedroom2         23.573457
Bathroom         23.599277
Car              25.039447
Landsize         33.881286
BuildingArea     60.576068
Propertycount     0.008607
dtype: float64
```

In [24]:

```
# The heatmap shows which variables are correlated.
melbourne_housing_df_heat = melbourne_housing_df.corr()
sns.heatmap(melbourne_housing_df_heat, annot=True, cmap='coolwarm')
```

Out[24]:

```
<Axes: >
```



Examining the heatmap shows that some variables are highly correlated with other variables. That might indicate the information is redundant in the dataset and could be a candidate for removal. For example, Rooms and Bedroom2. Bedroom2 is also missing a high percentage of its rows.

Similarly, features with very low correlation to the target variable 'price' are candidates for removal, like Landsize and Propertycount. BuildingArea has 60% of its rows missing and at 0.1 for its correlation to price, this features can also be removed.

The next batch of data cleaning addresses how to compensate for missing values. For example, some variables can be filled with a mean value (like Car), or remove the row if there are very few missing (like Distance), and keep highly correlated variables as is by just removing missing valued rows (like Bathroom). This is done in the data frame as follows.

In [26]:
```
del melbourne_housing_df['Bedroom2']
del melbourne_housing_df['Landsize']
del melbourne_housing_df['Propertycount']
del melbourne_housing_df['BuildingArea']
melbourne_housing_df['Car'].fillna(melbourne_housing_df['Car'].mean(), inplace=True)
melbourne_housing_df.dropna(axis=0, how='any', subset=None, inplace=True)
# finally see what quantity of rows and columns remain by checking the shape
melbourne_housing_df.shape
```

Out[26]:
```
(20800, 5)
```

Experimental Results

Now we are ready to train the linear regression model and make predictions. To aide this activity, a small function is used in training the model. Additionally, the model is tested on some new data to confirm it is functioning correctly, and then the model's predictions are evaluated based on applying it to the test data. Small functions can be used to help reuse the code that does the same steps for other models later.

In [29]:
```
# Define a function that takes a model as input along with the extracted
# data from the specific dataset.  This data is stored into variables that represent
# the feature matrix and target variable vector.
def dump_intercepts(model):
  print(f'the model intercepts are: {model.intercept_}')
def dump_coefficients(model, X):
  #model.coef_
  # now use them to display the data better, examine coefficients
  model_results = pd.DataFrame(model.coef_, X.columns, columns=['Coefficients'])
  print(f'the results of the model coefficients are:\n {model_results}')

# define a method to be used for doing the predictions
def do_predictions_linear(model, X_test, y_test):
  predicition = model.predict(X_test)
  mae = metrics.mean_absolute_error(y_test, predicition)
  print(f'the mean absolute error is: {mae}')

# This function uses a 70% training data subset and 30% test data subset
def execute_model(X, y, model):
```

```python
    # Note the use of a constant for the random_state to allow using the same
    # seed to the split operation.  Shuffle is also set to true to allow the
    # selection of data points randomly from the full set of data points.
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10, shuffle=True)
    model.fit(X_train, y_train)
    # Find the intercepts and coefficients
    dump_intercepts(model)
    dump_coefficients(model, X)

    # now attempt to do the predition on the test set.
    do_predictions_linear(model, X_test, y_test)
    # finally return the model to be used in other operations
    return model
```

In [30]:

```python
from sklearn.linear_model import LinearRegression
# Pull all the data together for passing into the functions
X = melbourne_housing_df[['Rooms', 'Distance', 'Bathroom', 'Car']]
y = melbourne_housing_df['Price']
# supply the algorithm to use
# save off the returned model trained for use later
mdl = execute_model(X, y, LinearRegression())
```

```
the model intercepts are: 282725.3156777753
the results of the model coefficients are:
            Coefficients
Rooms      269450.107900
Distance   -37787.766224
Bathroom   207173.059271
Car         47417.171595
the mean absolute error is: 363782.94232363265
```

In [31]:

```python
# Create a new house (not passed into the training set earlier) and predict its value.
# This is with the four inputs ordered as "Rooms", "Distance", "Bathroom", "Car" respectively
new_house = [[2, 2.5, 1, 1,]]
new_house_prediction = mdl.predict(new_house)
# examine the price
print(f'predicted house price {new_house_prediction}')
```

```
predicted house price [981746.34678379]
```

```
/home/ap/anaconda3/lib/python3.10/site-packages/sklearn/base.py:420: UserWarning: X does not have valid fea
ture names, but LinearRegression was fitted with feature names
  warnings.warn(
```

Running this series of functions shows a number of interesting operations. Firstly, most of the actions with the model were captured into generic functions. This is primarily to be able to reuse these functions and the operations they incorporate when needed for other models. Then after the training and operation, the model's predictive capabilities are evaluated by using the test set to determine the error. These are the most basic operations to conduct and are the first step to take. Lastly a new data point representing a new house example is created and fed into the model for a prediction. Based on the overall training, this new house example yields a target price value.

Some observations of the behavior of the model include a few odd issues. The mean absolute error is a big number. Coming in at approximately $364K, means that this is the amount by which the price of the home is in error. This is partly due to having removed so many variables from the dataframe when cleaning and also for having removed a number of rows. One critical variable that was removed was the "Type" of the home. Depending on if the example was an actual home or apartment, plays a big factor in determining the sale price.

The impact of this is that the predicited value of the home based on the four parameters provided yields a value of apporoximatley 982K dollars. In terms of the real value of the home, it is listed in the larger dataset at 1480K dollars. This is in the window of prices caused by the mean error given.

Logistic Regression

The purpose of this algorithm is to attempt to predict a discrete outcome (class or category). This set of classification problems can be for binary categories or multi-category. It can act on both continuous and discrete valued input features. The target varible it predicts is a discrete random value. It determines the probability of occurance of each class based on the training data. The distinguishing feature of logistics regression is the use of the Sigmoid function which converts that calculated probability into the class value.

Analysis

This example uses a dataset of the Kickstarter project campaigns. The purpose is to determine if a project will (or will not) reach its target funding level. This is a binary target random variable prediction problem. The dataset is imported into a Pandas dataframe and then cleaned up. This removes unnecessary variables, uses one-hot encoding to convert categorical features into numerics, and analyze the data for determining how to compensate for missing data.

In [37]:
```
kickstarter_df = pd.read_csv('18k_Projects.csv')
```

```python
# there are many pieces of data contained in it.
kickstarter_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18142 entries, 0 to 18141
Data columns (total 35 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   Id                                18142 non-null  int64  
 1   Name                              18142 non-null  object 
 2   Url                               18142 non-null  object 
 3   State                             18142 non-null  object 
 4   Currency                          18142 non-null  object 
 5   Top Category                      18142 non-null  object 
 6   Category                          18142 non-null  object 
 7   Creator                           18142 non-null  object 
 8   Location                          18142 non-null  object 
 9   Updates                           18142 non-null  int64  
 10  Comments                          18142 non-null  int64  
 11  Rewards                           18142 non-null  int64  
 12  Goal                              18142 non-null  int64  
 13  Pledged                           18142 non-null  int64  
 14  Backers                           18142 non-null  int64  
 15  Start                             18142 non-null  object 
 16  End                               18142 non-null  object 
 17  Duration in Days                  18142 non-null  int64  
 18  Facebook Connected                18142 non-null  object 
 19  Facebook Friends                  12290 non-null  float64
 20  Facebook Shares                   18142 non-null  int64  
 21  Has Video                         18142 non-null  object 
 22  Latitude                          9803 non-null   float64
 23  Longitude                         9803 non-null   float64
 24  Start Timestamp (UTC)             18142 non-null  object 
 25  End Timestamp (UTC)               18142 non-null  object 
 26  Creator Bio                       18142 non-null  object 
 27  Creator Website                   11475 non-null  object 
 28  Creator - # Projects Created      18142 non-null  int64  
 29  Creator - # Projects Backed       13898 non-null  float64
 30  # Videos                          18041 non-null  float64
 31  # Images                          18142 non-null  int64  
 32  # Words (Description)             18142 non-null  int64  
 33  # Words (Risks and Challenges)    18041 non-null  float64
 34  # FAQs                            18142 non-null  int64  
dtypes: float64(6), int64(13), object(16)
memory usage: 4.8+ MB
```

```
/tmp/ipykernel_3422/2186116625.py:1: DtypeWarning: Columns (27) have mixed types. Specify dtype option on i
mport or set low_memory=False.
  kickstarter_df = pd.read_csv('18k_Projects.csv')
```

In [38]:

```python
# Examine the data stored
kickstarter_df.head()
```

Out[38]:

	Id	Name	Url	State	Currency	Top Category	Category	Creator	Location	Updates	...	Tim
0	1007121454	Nail Art and Photos Printed on your Nails w/ E...	https://www.kickstarter.com/projects/137019948...	failed	USD	Art	Art	Dodie Egolf	Puyallup	0	...	201 01:5.
1	2032015036	Cold Again	https://www.kickstarter.com/projects/737783165...	failed	USD	Film & Video	Short Film	James Jacobs	Boston	0	...	201 02:3
2	733782855	Uchu Bijin Jewelry	https://www.kickstarter.com/projects/uchubijin...	failed	USD	Fashion	Fashion	Uchu Bijin	New York	1	...	201 01:2
3	514687871	Poetically Speaking: Stories of Love, Triumph ...	https://www.kickstarter.com/projects/tylicee/p...	failed	USD	Publishing	Poetry	Tylicee Mysreign	Detroit	0	...	201 01:1
4	683545993	Stranger Travels: Teachings from the Heart of ...	https://www.kickstarter.com/projects/197270300...	failed	USD	Publishing	Nonfiction	Ian Driscoll	Pucallpa	0	...	201 01:1

5 rows × 35 columns

In [39]:
```
# data is removed that is not needed
del kickstarter_df['Id']
del kickstarter_df['Name']
del kickstarter_df['Url']
del kickstarter_df['Location']
del kickstarter_df['Pledged']
del kickstarter_df['Creator']
del kickstarter_df['Category']
del kickstarter_df['Updates']
del kickstarter_df['Start']
del kickstarter_df['End']
del kickstarter_df['Latitude']
del kickstarter_df['Longitude']
del kickstarter_df['Start Timestamp (UTC)']
del kickstarter_df['End Timestamp (UTC)']
del kickstarter_df['Creator Bio']
del kickstarter_df['Creator Website']
# Use one-hot encoding
kickstarter_df = pd.get_dummies(kickstarter_df, columns=['State', 'Currency', 'Top Category', 'Facebook Cor
# now compare and see how much data is left
kickstarter_df.shape
```

Out[39]:
```
(18142, 36)
```

In [40]:
```
# Now its time to compensate for missing data by first determining how many there are.
kickstarter_df.isnull().sum()
```

Out[40]:

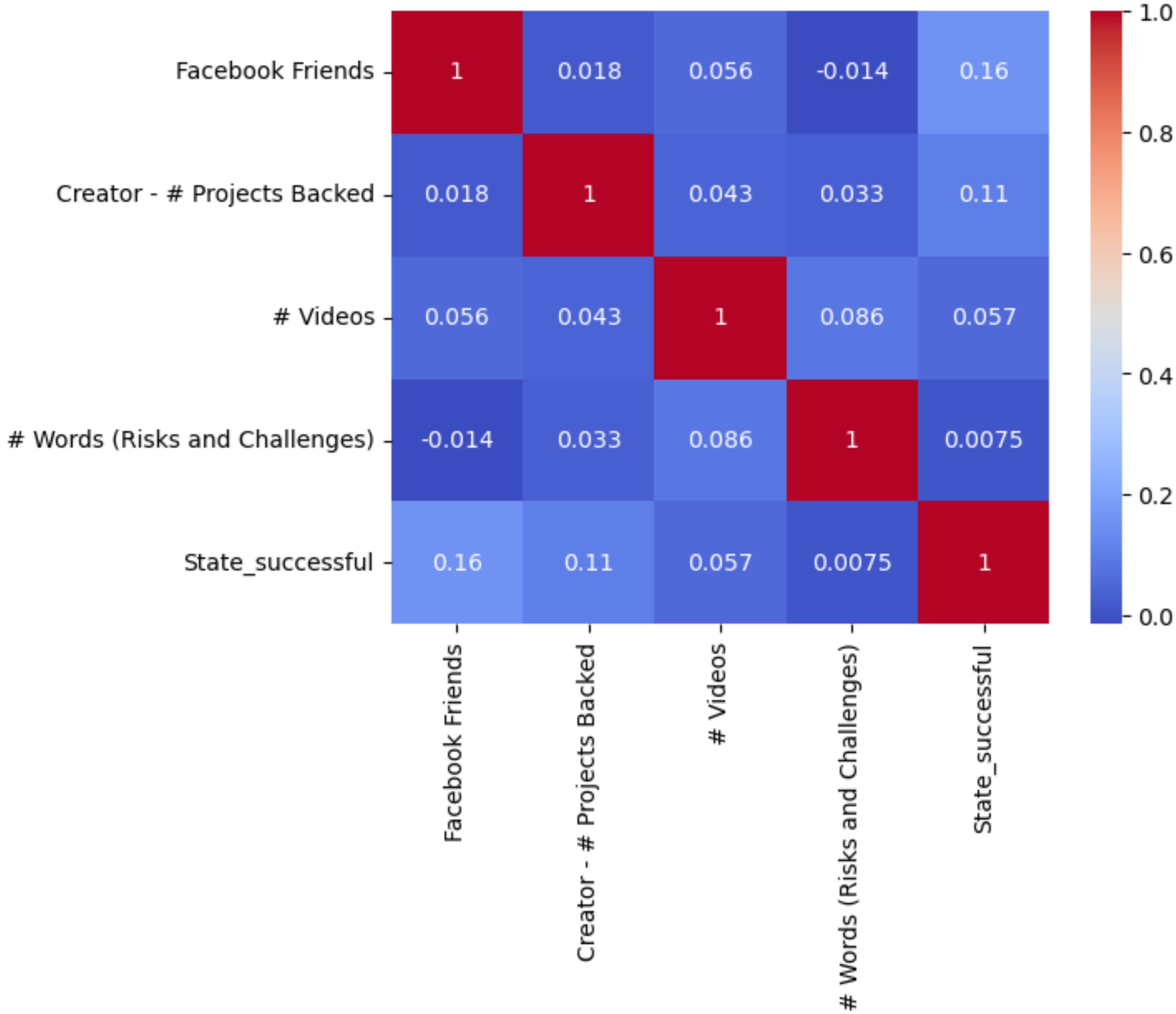
```
Comments                             0
Rewards                              0
Goal                                 0
Backers                              0
Duration in Days                     0
Facebook Friends                  5852
Facebook Shares                      0
Creator - # Projects Created         0
Creator - # Projects Backed       4244
# Videos                           101
# Images                             0
# Words (Description)                0
# Words (Risks and Challenges)     101
# FAQs                               0
State_successful                     0
Currency_CAD                         0
Currency_EUR                         0
Currency_GBP                         0
Currency_NZD                         0
Currency_USD                         0
Top Category_Comics                  0
Top Category_Crafts                  0
Top Category_Dance                   0
Top Category_Design                  0
Top Category_Fashion                 0
Top Category_Film & Video            0
Top Category_Food                    0
Top Category_Games                   0
Top Category_Journalism              0
Top Category_Music                   0
Top Category_Photography             0
Top Category_Publishing              0
Top Category_Technology              0
Top Category_Theater                 0
Facebook Connected_Yes               0
Has Video_Yes                        0
dtype: int64
```

In [41]:

```
# Since four features have missing values.  Determine how correlated they are to the target variable State_
kickstarter_coor_df = kickstarter_df[['Facebook Friends', 'Creator - # Projects Backed', '# Videos', '# Wo
# Display a heat map and examine the correlations
kickstarter_coor_df_heat = kickstarter_coor_df.corr()
```

```
sns.heatmap(kickstarter_coor_df_heat, annot=True, cmap='coolwarm')
```

Out[41]: <Axes: >



These four inputs features have missing values that must be mitigated. For the feature variables that are highly correlated to the target variable State_successful, some action must be taken. For others, either the missing rows or feature variable can be removed. The variables for # Videos and # Words have very few missing rows and are not highly correlated to the target. So in both of these cases, the missing rows can be removed. For the other two, their correlation is much higher so they must be examined to determine what strategy can be used to account for the missing rows. The variance, range, and standard deviation can be examined to see what the data looks like.

In [43]:
```
# examine descriptive statistics
kickstarter_df.describe()
```

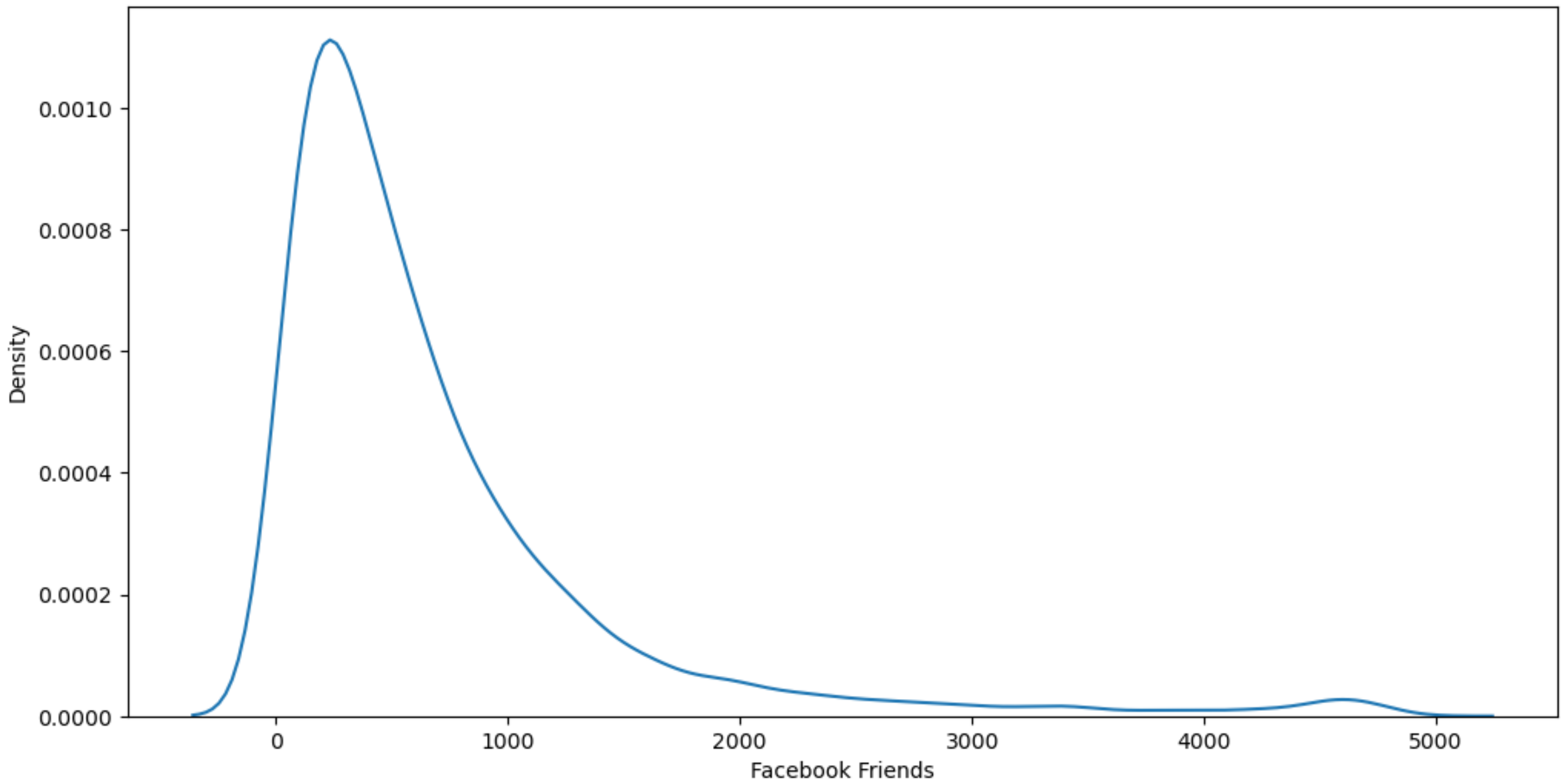
Out[43]:

	Comments	Rewards	Goal	Backers	Duration in Days	Facebook Friends	Facebook Shares	Creator - # Projects Created
count	18142.000000	18142.000000	1.814200e+04	18142.000000	18142.000000	12290.000000	18142.000000	18142.000000
mean	34.243027	10.042002	2.653121e+04	138.070279	31.398468	694.233686	396.729137	1.520119
std	539.161283	5.889806	7.583874e+05	633.787780	10.058827	783.802343	2544.711314	2.540474
min	0.000000	2.000000	1.000000e+02	1.000000	1.000000	0.000000	0.000000	1.000000
25%	0.000000	6.000000	2.000000e+03	7.000000	29.000000	216.250000	21.000000	1.000000
50%	0.000000	9.000000	5.000000e+03	29.000000	30.000000	453.000000	104.000000	1.000000
75%	3.000000	12.000000	1.500000e+04	89.000000	32.000000	860.000000	322.000000	1.000000
max	30341.000000	131.000000	1.000000e+08	35383.000000	60.000000	4885.000000	260505.000000	111.000000

8 rows × 36 columns

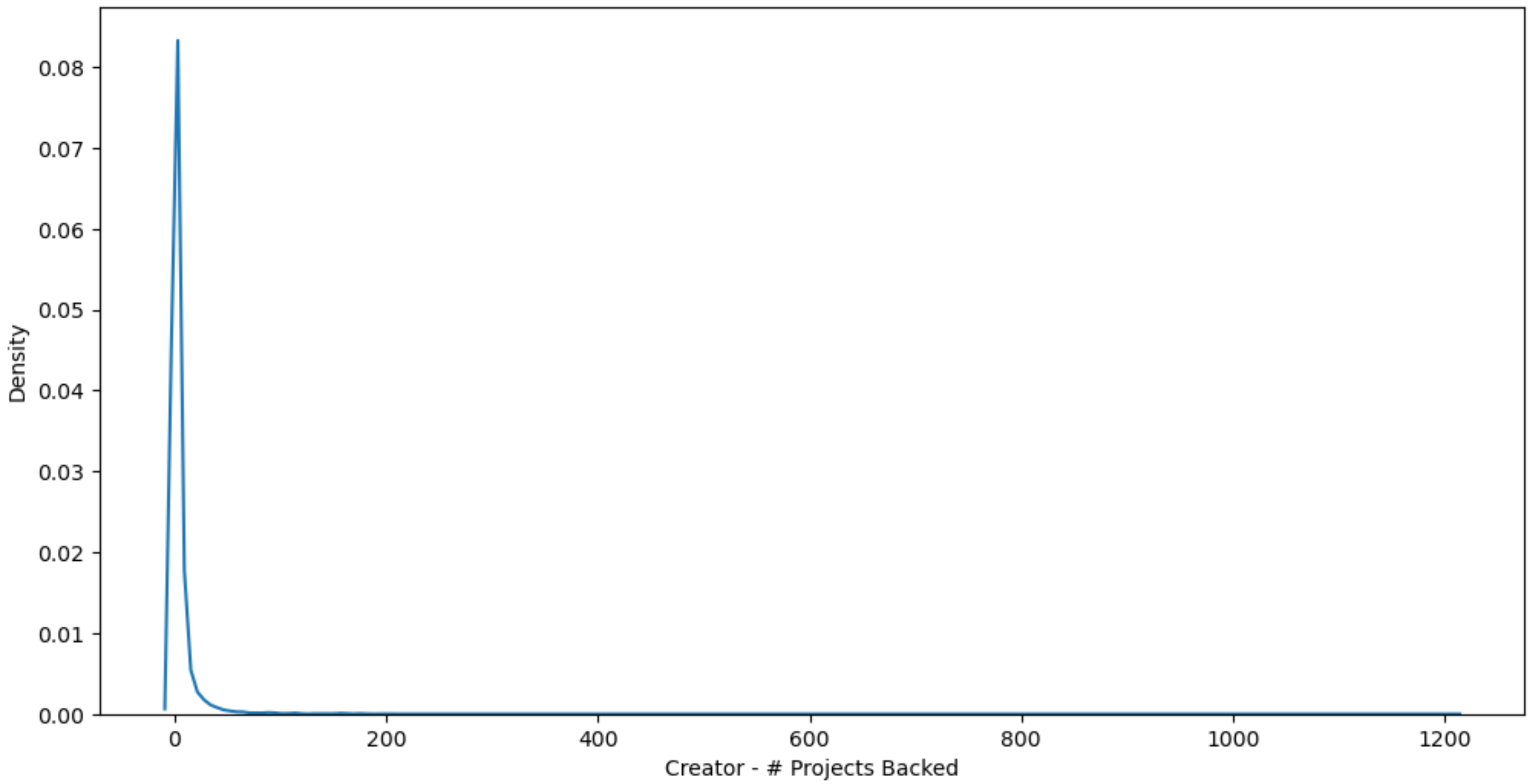
In [44]:
```
# examine variable distributions
plt.figure(figsize=(12,6))
sns.kdeplot(kickstarter_df['Facebook Friends'])
```

Out[44]: <Axes: xlabel='Facebook Friends', ylabel='Density'>



In [45]:
```
# examine variable distributions
plt.figure(figsize=(12,6))
sns.kdeplot(kickstarter_df['Creator - # Projects Backed'])
```

Out[45]: <Axes: xlabel='Creator - # Projects Backed', ylabel='Density'>



The first figure show that there is high variance for 'Facebook Friends' feature. The other descriptive statistics are not good candidates for substitution into the missing rows. The feature is correlated with the target, so it is best to retain the feature and remove its missing rows. The second figure shows 'Creator - # Projects Backed' has similar issues, but its range and more importantly, its standard deviation is much smaller. This feature's missing rows are a candidate for substitution with the mean.

In [47]:
```
# Clean up remaining data
kickstarter_df['Creator - # Projects Backed'].fillna(kickstarter_df['Creator - # Projects Backed'].mean(),
# remove all the other rows
kickstarter_df.dropna(axis=0, how='any', subset=None, inplace=True)
# examine how much data is left, showing there is a decent amount of data to still train the model with.
kickstarter_df.shape
```

Out[47]:
```
(12215, 36)
```

Experimental Results

Now we are ready to train the logistic regression model and make predictions. Nearly identical steps are taken as in the previous section for creating the outputs desired. In the case of logistic regression being used on a classification problem, the evaluation measures are not mean absolute error. Instead it is the confusion matrix and classification report.

In [50]:
```
# define some helper functions
def execute_model_logistic(X, y, model):
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10, shuffle=True)
  model.fit(X_train, y_train)
  # now see how it performed
  model_predict = model.predict(X_test)
  # evaluate it with measures
  print(confusion_matrix(y_test, model_predict))
  print(classification_report(y_test, model_predict))
  return model
```

In [51]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
X = kickstarter_df.drop('State_successful', axis=1)
y = kickstarter_df['State_successful']
mdl = execute_model_logistic(X, y, LogisticRegression())
```

```
[[1659  170]
 [ 229 1607]]
              precision    recall  f1-score   support

           0       0.88      0.91      0.89      1829
           1       0.90      0.88      0.89      1836

    accuracy                           0.89      3665
   macro avg       0.89      0.89      0.89      3665
weighted avg       0.89      0.89      0.89      3665
```

```
/home/ap/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

The overall performance of the train and test data is shown in the two reports. The confusion matrix shows 170 false-negatives and 229 false-positives. This is a good result for the quantity of entries. Additionally, the classification report shows precision and recall that are approximately 90% and 88% respectively. This is also a good result with no optimizations having been applied other than data cleaning. Lastly, the new project prediction is positive indicating a successful project.

Support Vector Machines

The theory section earlier detailed how SVMs work. The following sections will demonstrate the use of SVM as a binary classifier. Support Vector Classifier is the algorithm to be used.

Analysis

A new dataset used here is the advertising dataset. It is a labelled dataset that has a few independent features related to the demographics of the user and how much time is being spent on a website. The target variable indicates if the user did or did not click on an advertisement. The SVC will be trained to predict the target variable and the confusion matrix and classification report used to evaluate the performance of the classification predictions on the test data set.

As before, the dataset is imported and cleaned before being used for training of the model. Since the dataset has a very small set of features, the cleaning is minimal. Some variables are removed and others one-hot encoded.

In [57]:
```
advertising_df = pd.read_csv('advertising.csv')
advertising_df.head()
```

Out[57]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11	0
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02	0
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42	0
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19	0
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18	0

In [58]:
```
advertising_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Daily Time Spent on Site  1000 non-null   float64
 1   Age                       1000 non-null   int64  
 2   Area Income               1000 non-null   float64
 3   Daily Internet Usage      1000 non-null   float64
 4   Ad Topic Line             1000 non-null   object 
 5   City                      1000 non-null   object 
 6   Male                      1000 non-null   int64  
 7   Country                   1000 non-null   object 
 8   Timestamp                 1000 non-null   object 
 9   Clicked on Ad             1000 non-null   int64  
dtypes: float64(3), int64(3), object(4)
memory usage: 78.2+ KB
```

In [59]:
```
advertising_df.describe()
```

Out[59]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Clicked on Ad
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000
mean	65.000200	36.009000	55000.000080	180.000100	0.481000	0.50000
std	15.853615	8.785562	13414.634022	43.902339	0.499889	0.50025
min	32.600000	19.000000	13996.500000	104.780000	0.000000	0.00000
25%	51.360000	29.000000	47031.802500	138.830000	0.000000	0.00000
50%	68.215000	35.000000	57012.300000	183.130000	0.000000	0.50000
75%	78.547500	42.000000	65470.635000	218.792500	1.000000	1.00000
max	91.430000	61.000000	79484.800000	269.960000	1.000000	1.00000

In [60]:

```
del advertising_df['Ad Topic Line']
del advertising_df['Timestamp']
advertising_df = pd.get_dummies(advertising_df, columns=['Country', 'City'])
advertising_df.head()
```

Out[60]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Clicked on Ad	Country_Afghanistan	Country_Albania	Country_Algeria	Country_American Samoa
0	68.95	35	61833.90	256.09	0	0	0	0	0	0
1	80.23	31	68441.85	193.77	1	0	0	0	0	0
2	69.47	26	59785.94	236.50	0	0	0	0	0	0
3	74.15	29	54806.18	245.89	1	0	0	0	0	0
4	68.37	35	73889.99	225.58	0	0	0	0	0	0

5 rows × 1212 columns

Experimental Results

Now the dataframe has been cleaned and is ready to be used in training. Just as before, some support functions will be used to help with the process. Also, an optimization step will be used by finding better model hyperparameters to use and seeing if it improves prediction metrics.

In [63]:
```
# define the function
def evaluate_model_svc(X, y, model):
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)
  model.fit(X_train, y_train)
  model_predict = model.predict(X_test)
  print(confusion_matrix(y_test, model_predict))
  print(classification_report(y_test, model_predict))
  return model
```

In [64]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
X = advertising_df.drop('Clicked on Ad', axis=1)
y = advertising_df['Clicked on Ad']
mdl = evaluate_model_svc(X, y, SVC())
```

```
[[124  22]
 [ 68  86]]
              precision    recall  f1-score   support

           0       0.65      0.85      0.73       146
           1       0.80      0.56      0.66       154

    accuracy                           0.70       300
   macro avg       0.72      0.70      0.70       300
weighted avg       0.72      0.70      0.69       300
```

These reports show the overall quality of the predictions. The false-positives is 68 and the recall is just 56%. Not very good due to the high occurance of false-positives. The precision is better at 80% but the f1-score is not great. The preformance can be improved using an optimization step of grid search.

The SVC algorithm has many hyperparameters to adjust, but only C and gamma are used here. Paraphrasing from the description given in [5]: "The hyperparameter C controls the cost of misclassification on the training data." and "Gamma refers to the Gaussian radial basis function and the influence of the support vector."

What these two definitions are saying is that C affects the margin (the distinguishing feature of SVM). The margin is considered 'soft' as adjusted by the value of C. C will control how many mis-classification errors are ignored, such that no penalty is applied. This affects the training in which data points that cross the margin boundaries are or are not allowed to affect the training process (that is attempting to define the margin). The Gamma, on the other hand, impacts a model's bias and variance. Small gamma causes high bias and low variance, large gamma causes low bias and high variance.

As with grid search used before, the operation takes a range of hyperparameter values that will be the set of combinations to be tested. The output of the model fit is the best hyperparameter combination to use. This optimized set is then used to do the predictions and classification measures examined for improvement.

In [66]:

```
# helper function to assist the training and evaluation
def evaluate_model_svc_gs(X, y, model, hyperparams):
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)
  grid = GridSearchCV(model, hyperparams)
  grid.fit(X_train, y_train)
  # the fit will find the optimized values
  print(f'best parameters found: {grid.best_params_}')
  grid_predict = grid.predict(X_test)
  print(confusion_matrix(y_test, grid_predict))
  print(classification_report(y_test, grid_predict))
  return model
```

In [67]:

```
# now attempt to search through the supplied parameters and use the optimized values.
hyperparameters = {'C':[10,25,50], 'gamma':[0.001, 0.0001, 0.00001]}
mdl = evaluate_model_svc_gs(X, y, SVC(), hyperparameters)
```

```
best parameters found: {'C': 50, 'gamma': 1e-05}
[[129  17]
 [ 15 139]]
              precision    recall  f1-score   support

           0       0.90      0.88      0.89       146
           1       0.89      0.90      0.90       154

    accuracy                           0.89       300
   macro avg       0.89      0.89      0.89       300
weighted avg       0.89      0.89      0.89       300
```

The overall performance of the train and test data is shown in the two reports. The confusion matrix shows 17 false-negatives and 15 false-positives. This is an improvement from before. Additionally, the classification report shows precision and recall that are approximately 89% and 90% respectively. This is the result of optimizations having been applied other than data cleaning alone.

k-Nearest Neighbors (KNN)

KNN is another familiar algorithm applied here to the advertising dataset. This algorithm is best suited to small datasets due to its large memory needs for storing all the data points. Data reduction techniques and data scaling techniques are used to reduce data set size. This cleaned data is then used for training.

Analysis

In [72]:
```
# The same advertising dataset is used here, but this time it is greatly reduced.
advertising_df2 = pd.read_csv('advertising.csv')
del advertising_df2['Ad Topic Line']
del advertising_df2['Timestamp']
del advertising_df2['Male']
del advertising_df2['Country']
del advertising_df2['City']
# Now the data is scaled into a new dataframe
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(advertising_df2.drop('Clicked on Ad', axis=1))
scaled_features = scaler.transform(advertising_df2.drop('Clicked on Ad', axis=1))
# The purpose of doing the scaling is to create a new dataset frame with features that have standarized va
# This gives all features equal weighting in that no single feature with large variance, range, standard de
```

Experimental Results

Now the model is trained on the cleaned data set first. Then the hyperparameter of the number of neighbors k is selected to see if produces a better result. These two will attempt to see what the impact of using an optimized parameter is on the prediction performance.

In [75]:
```
# define a helper function to use
def evaluate_model_knn(X, y, model):
```

```
    print(f'KNN model with neighbor count k = {model.n_neighbors}')
    # use the passed in data to do the split and training
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10, shuffle=True)
    model.fit(X_train, y_train)
    # evaluation on the test set is done and then the confusion matrix and classification report displayed
    model_predict = model.predict(X_test)
    print(confusion_matrix(y_test, model_predict))
    print(classification_report(y_test, model_predict))
    return model
```

In [76]:
```
from sklearn.neighbors import KNeighborsClassifier
# Run the first iteration of training with k=5 neighbors
X = scaled_features
y = advertising_df2['Clicked on Ad']
mdlA = evaluate_model_knn(X, y, KNeighborsClassifier(n_neighbors=5))

# Now try with set of k values to search for which one gives the best result
def search_knn_value(X, y):
  for k in [3,4,7,15,25,31]:
    mdl = evaluate_model_knn(X, y, KNeighborsClassifier(n_neighbors=k))
# end
# run the optimization by searching for the right hyperparameter value
search_knn_value(X,y)
```

```
KNN model with neighbor count k = 5
[[144   2]
 [ 10 144]]
              precision    recall  f1-score   support

           0       0.94      0.99      0.96       146
           1       0.99      0.94      0.96       154

    accuracy                           0.96       300
   macro avg       0.96      0.96      0.96       300
weighted avg       0.96      0.96      0.96       300

KNN model with neighbor count k = 3
[[143   3]
 [  9 145]]
              precision    recall  f1-score   support

           0       0.94      0.98      0.96       146
           1       0.98      0.94      0.96       154

    accuracy                           0.96       300
   macro avg       0.96      0.96      0.96       300
weighted avg       0.96      0.96      0.96       300

KNN model with neighbor count k = 4
[[144   2]
 [ 11 143]]
              precision    recall  f1-score   support

           0       0.93      0.99      0.96       146
           1       0.99      0.93      0.96       154

    accuracy                           0.96       300
   macro avg       0.96      0.96      0.96       300
weighted avg       0.96      0.96      0.96       300

KNN model with neighbor count k = 7
[[143   3]
 [ 10 144]]
              precision    recall  f1-score   support

           0       0.93      0.98      0.96       146
```

```
           1       0.98      0.94      0.96       154

    accuracy                           0.96       300
   macro avg       0.96      0.96      0.96       300
weighted avg       0.96      0.96      0.96       300

KNN model with neighbor count k = 15
[[143   3]
 [ 11 143]]
              precision    recall  f1-score   support

           0       0.93      0.98      0.95       146
           1       0.98      0.93      0.95       154

    accuracy                           0.95       300
   macro avg       0.95      0.95      0.95       300
weighted avg       0.95      0.95      0.95       300

KNN model with neighbor count k = 25
[[144   2]
 [ 12 142]]
              precision    recall  f1-score   support

           0       0.92      0.99      0.95       146
           1       0.99      0.92      0.95       154

    accuracy                           0.95       300
   macro avg       0.95      0.95      0.95       300
weighted avg       0.96      0.95      0.95       300

KNN model with neighbor count k = 31
[[144   2]
 [ 13 141]]
              precision    recall  f1-score   support

           0       0.92      0.99      0.95       146
           1       0.99      0.92      0.95       154

    accuracy                           0.95       300
   macro avg       0.95      0.95      0.95       300
weighted avg       0.95      0.95      0.95       300
```

In [77]:
```python
# finally evaluate it with k=3 the optimal value and observe click count
mdlB = evaluate_model_knn(X, y, KNeighborsClassifier(n_neighbors=3))
# and attempt to predict how many ad clicks occur in a subset of the data
print(mdlB.predict(scaled_features)[0:10])
```

```
KNN model with neighbor count k = 3
[[143   3]
 [  9 145]]
              precision    recall  f1-score   support

           0       0.94      0.98      0.96       146
           1       0.98      0.94      0.96       154

    accuracy                           0.96       300
   macro avg       0.96      0.96      0.96       300
weighted avg       0.96      0.96      0.96       300

[0 0 0 0 0 0 0 1 0 0]
```

As shown in the confusion matrix and classification report, the quality of predictions improves with the k set to 3. The false positives improves by 1 (from 10 down to 9) as compared with k=5. The other numbers showed increasing false-negatives. The other metrics are mostly stable. The benefit of confirming this model can function with a lower value of k is that it reduces training time and compute resources required. The scaled features subset of data shows the expected number of clicks, indicating one user clicked through on the ad.

Decision Trees (DT)

As discussed before, decision trees and random forests (an ensemble methodology) are bagging approaches. These function in parallel and independently of each other. DT can operate on both discrete and continuous features. They can also predict continuous target variables (regressors) or discrete target variables (classifiers).

DT use a splitting technique and is based on the variance or entropy of the feature data that contributes to each class or category being predicted. It can suffer from overfitting.

Analysis

The DT will be shown to function over the same advertising dataset for comparison purposes in their classification role. As before, the data is first cleaned up to prepare it for prediction. Then it is used to train a DT classifier to determine its prediction capability.

In [83]:
```
advertising_df_dt = pd.read_csv('advertising.csv')
del advertising_df_dt['Ad Topic Line']
del advertising_df_dt['Timestamp']
advertising_df_dt = pd.get_dummies(advertising_df_dt, columns=['Country', 'City'])
advertising_df.head()
```

Out[83]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Clicked on Ad	Country_Afghanistan	Country_Albania	Country_Algeria	Country_American Samoa
0	68.95	35	61833.90	256.09	0	0	0	0	0	0
1	80.23	31	68441.85	193.77	1	0	0	0	0	0
2	69.47	26	59785.94	236.50	0	0	0	0	0	0
3	74.15	29	54806.18	245.89	1	0	0	0	0	0
4	68.37	35	73889.99	225.58	0	0	0	0	0	0

5 rows × 1212 columns

Experimental Results

In [85]:
```
# Create a helper function to use in the evaluation
def evaluate_model_dt(X, y, model):
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10, shuffle=True)
  model.fit(X_train, y_train)
  model_predict = model.predict(X_test)
  print(confusion_matrix(y_test, model_predict))
  print(classification_report(y_test, model_predict))
  return model
```

In [86]:
```
from sklearn.tree import DecisionTreeClassifier
X = advertising_df_dt.drop('Clicked on Ad', axis=1)
y = advertising_df_dt['Clicked on Ad']
```

```
# Run the model
mdl = evaluate_model_dt(X, y, DecisionTreeClassifier())
```

```
[[139   7]
 [  8 146]]
              precision    recall  f1-score   support

           0       0.95      0.95      0.95       146
           1       0.95      0.95      0.95       154

    accuracy                           0.95       300
   macro avg       0.95      0.95      0.95       300
weighted avg       0.95      0.95      0.95       300
```

The output in the confusion matrix shows only 9 false-positives and 94% in precision and recall scores.

Random Forests (RF)

RF are collections of DT that are training to combat the overfitting problem. The outcome of the prediction of the target variable is done by class voting (or averging for regression). The RF uses subsets of features randomly selected to train each DT. This randomization helps reduce overfitting but also produces a better result than a single DT alone.

Analysis

The same advertising dataset is used again for comparison with previously trained models and outcomes. It is similarly cleaned up as the DT, so there is no need to repeat those steps here.

Experimental Results

In [94]:
```
# The same helper function from DT can be used here because the internal steps are identical.
# Only the algorithm used changes and is passed into the function.
from sklearn.ensemble import RandomForestClassifier
# use the same X, y
# Run the model
mdl = evaluate_model_dt(X, y, RandomForestClassifier())
```

```
[[140   6]
 [  7 147]]
              precision    recall  f1-score   support

           0       0.95      0.96      0.96       146
           1       0.96      0.95      0.96       154

    accuracy                           0.96       300
   macro avg       0.96      0.96      0.96       300
weighted avg       0.96      0.96      0.96       300
```

The predictions improve by having only 7 false-positives and 6 false-negatives. Also the precision goes up to 97% and recall to 95%. The f-1 score is now at 96%.

Gradient Boosting

As shown with the bagging random forest method, the boosting variants are also ensemble methods. The difference being that the boosting approach is not done in parallel but in serial. This serial approach allows for correcting errors made in earlier model runs during later model runs by having the latter models learn from earlier results. It can be used for either regression or classification.

This approach improves upon DT and RF by using a technique that examines the weakly performing models and uses a weighting factor that controls the contribution of those weak models in later rounds. Also, earlier strongly performing models are replaced with weaker models. This allows the replacements to learn from earlier mistakes to improve performance.

Analysis

The same advertising dataset is used again for comparison with previously trained models and outcomes. It is similarly cleaned up as the DT, so there is no need to repeat those steps here.

Experimental Results

In [100…

```
# The same helper function from DT can be used here because the internal steps are identical.
# Only the algorithm used changes and is passed into the function.
from sklearn import ensemble
# use the same X, y
```

```
# Run the model with a set of hyperparameters to optimize for performance
m = ensemble.GradientBoostingClassifier(n_estimators=250, learning_rate=0.1, max_depth=5, min_samples_split
                                        min_samples_leaf=6, max_features=0.6, loss = 'deviance')
mdl = evaluate_model_dt(X, y, m)
```

```
/home/ap/anaconda3/lib/python3.10/site-packages/sklearn/ensemble/_gb.py:280: FutureWarning: The loss parame
ter name 'deviance' was deprecated in v1.1 and will be removed in version 1.3. Use the new parameter name '
log_loss' which is equivalent.
  warnings.warn(
[[141   5]
 [  8 146]]
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       146
           1       0.97      0.95      0.96       154

    accuracy                           0.96       300
   macro avg       0.96      0.96      0.96       300
weighted avg       0.96      0.96      0.96       300
```

The predictions improve by having the 8 false-positives, but only 5 false-negatives. Also the precision is the same at 97% and recall at 95%. The f-1 score is at 96%.

Conclusions

This article has covered a lot of material. Most of the algorigthms shown were the same as seen in previous articles. The one new algorithm was SVM. Its unique margin feature as a mechanism for improving binary classification makes it robust to outliers influencing the predictions.

The main aspects of this review were to see regression and classification using new datasets not used before. These new datasets have unique features and they each had to be cleaned appropriately before attempting to train models. Many of the datasets had many more features than required or had many missing examples. The former were candidates for removal, especially if they were weakly correlated to the target variable. The latter had to use approches to compensate for missing data or just remove those rows if they did not have a negative impact on training.

Lastly, the advertising dataset was used across a number of different algorithms. This showed a common dataset theme applied to a handful of algorithms to compare how each one improves the quality of predicitive performance. There was minimal hyperparameter tuning done across all these algorithms, so the step up in performance could easily be seen between them.

References

[1] https://ap20.github.io/nnj/NL/edutechrev/IntroMLPart1_regression_apatel.html

[2] https://ap20.github.io/nnj/NL/edutechrev/IntroMLPart2_classification_apatel.html

[3] https://ap20.github.io/nnj/NL/edutechrev/ExploreDAandVisPart1_apatel.html

[4] https://ap20.github.io/nnj/NL/edutechrev/EDAandVisPart2_apatel.html

[5] Theobald, Oliver. Machine Learning with Python, 2024. scatterplotspress.com. https://scatterplotpress.teachable.com/

[6] https://scikit-learn.org/stable/modules/svm.html

[7] https://www.geeksforgeeks.org/support-vector-machine-algorithm/

[8] https://medium.com/low-code-for-advanced-data-science/support-vector-machines-svm-an-intuitive-explanation-b084d6238106