

Homework 4 (100 points) (Regression & Classification)

This assignment has 3 separate problems. Each problem will require you to create a separate source code pipeline (using a jupyter notebook) and perform all the requisite analyses. In addition to performing the analysis (i.e., writing code in the jupyter notebook), any questions that have been posed in the word document can be answered directly inline (i.e., below each question in the document itself). The assignment is worth 100 points in total.

What to submit:

1. For each question, you will need to create a separate jupyter notebook with all the requisite code / analysis.
2. In addition, you can use this word document to answer all the questions (inline) and submit a single document for all the analysis.

Submission format: Do not modify the directory structure of the zip archive that you download to start the assignment. Essentially, make sure each jupyter notebook is in a separate directory (which is how the assignment is initially provided i.e., with Q1, Q2, Q3 subdirectories). The *HW4_Questions_and_solutions.pdf* (i.e., the current file with the solutions that you will fill in and convert to PDF format), should be placed at the root of the zip archive.

Once all the solution pdf and all the source code is complete, zip the full assignment and submit the zip archive. Your submission zip archive should have the following structure:

```
<lastname_firstname_CS556B_HW4.zip>/.
  /HW4_Questions_and_Solutions.pdf
  Q1/q1.ipynb
  Q2/q2.ipynb
  Q3/q3.ipynb
```

Leave the datasets unchanged and in the directory itself. Each of your jupyter notebooks should be directly executable without any further modifications (i.e., if your assignment submission is downloaded from canvas and unzipped), any of your submitted notebooks if executed using a jupyter server should be directly runnable without additional data downloads etc.

SUBMISSION BY
SANJEET VINOD JAIN
CWID 20012768

Q1. Housing Price Prediction (34 points)

- a. **Dataset Description:** The data pertains to the houses found in each California district and some summary statistics about them based on the 1990 census data. It contains one instance per district block group. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people). The goal of this task is to design a regression model to predict the *median house value* conditioned upon a set of input attributes corresponding to a particular California district block.

The attributes in the dataset are as follows; their names are self-explanatory:

- i. longitude (continuous): One of the coordinates that are used to identify the California district block
- ii. latitude (continuous): One of the coordinates that are used to identify the California district block
- iii. housing_median_age (continuous): Average age of the house in California district block
- iv. total_rooms (continuous): Total number of rooms of all the houses in the California district block
- v. total_bedrooms (continuous): Total number of bedrooms of all the houses in the California district block
- vi. population (continuous): Number of people residing in the district block
- vii. households (continuous): Number of families in the district block
- viii. median_income (continuous): Median income for households in the district block of houses (measured in tens of thousands of US Dollars)
- ix. ocean_proximity (categorical): Location of the house. Is it inland, near the bay, near the ocean, etc.
- x. median_house_value (continuous): Median house value within a district block (measured in US Dollars)

Our target variable will be median house value. Use the rest of the fields mentioned above to predict the median house value.

b. **Data Loading / Preprocessing: (10 points)**

- i. Loading: (2 points)
 1. Load the California housing dataset using [pandas.read_csv\(\)](#) function and store it in the variable (i.e., a pandas dataframe) named 'df'.
 2. The resulting data frame should have the shape (20640, 10) indicating that there are 20640 rows and 10 columns.
 3. Find the missing values in the data frame. If any (i.e., even if one column in each instance / row has a missing value), drop the row using [pandas.DataFrame.dropna\(\)](#) function. The resulting data frame should have the shape (20433, 10) indicating that there are 20433 rows and 10 columns.
 4. Create a data frame 'corr_df' by dropping the columns *latitude*, *longitude*, and *ocean_proximity* using the [pandas.DataFrame.drop\(\)](#) function. Use the *Pearson correlation* to find the correlation of each remaining feature in the 'corr_df' with the target variable '*median_house_value*' using the function [pandas.DataFrame.corrwith\(\)](#). Report results in the following table.

Feature Name	Pearson Correlation
housing_median_age	0.106432
total_rooms	0.133294
total_bedrooms	0.049686
population	-0.025300
households	0.064894
median_income	0.688355

5. Create a data frame X of features (by dropping the column '*median_house_value*' from the original data frame) using the [pandas.DataFrame.drop\(\)](#) function. Create a Series object of targets Y (by only considering the '*median_house_value*' column from the original data frame (Do NOT use the 'corr_df' data frame in this step. Use the data frame which was obtained as a result of step b.i.3 above).

ii. Data Visualization: (3 points)

1. Use [pandas.DataFrame.hist\(bins = 50\)](#) function for visualizing the variation on the columns *housing_median_age*, *total_rooms*, *total_bedrooms*, *population*, *household*, *median_income* and *median_house_value*. Plot each histogram as a separate subplot.
2. Use [pandas.dataframe.describe\(\)](#) function to find the mean, median and standard deviations for each feature and report in the jupyter notebook.
3. Use [pandas.get_dummies](#) to convert categorical variables into dummy /one-hot encoding. In this case the categorical column is *ocean_proximity*

iii. Data Splitting: (2 points)

1. Split data into training and test sets using the sklearn [train_test_split\(\)](#) function. Perform 70-30 distribution i.e. 70% training and 30% testing. The result of your data split should yield 4 separate data frames *X_train*, *X_test*, *y_train*, *y_test*. (respectively, the training features, testing features, training targets and testing target).

iv. Data Scaling: (3 points)

1. Use the [StandardScaler\(\)](#) to instantiate the standard scaler class. **Note:** You will need two separate scaler objects, one to scale the features, another to scale the target values.
2. For each scaler, employ the `'fit_transform()'` function (only on the training features, training targets) of the scaler to retrieve the new (scaled) version of the data. Store them in *X_train*, and *y_train* again.
3. Scale the *X_test* and *y_test* as well and store the scaled values back in *X_test* and *y_test*. (i.e., use the appropriate “fitted” scaler above to “transform” the test data. Note: the function to be employed in this case is `'transform()'` as opposed to `'fit_transform()'`). Henceforth, *X_train*, *y_train*, *X_test*, *y_test* will refer to the scaled data unless stated otherwise.
4. Use [pandas.DataFrame.hist\(bins = 50\)](#) function for visualizing the variation of numerical attributes *housing_median_age*, *total_rooms*, *total_bedrooms*, *population*, *household*, *median_income* and *median_house_value* for the *X_train* and *y_train* dataset (similar to step b.ii.1 above). Once again, plot each histogram as a separate subplot.

c. Modelling: (10 points)

- i. Employ Linear Regression from [sklearn.linear_model](#), and instantiate the model.
- ii. (3 points) Once instantiated, `'fit()'` the model using the *scaled X_train*, *y_train* data.
- iii. (2 points) Employ the `'predict()'` function to obtain predictions on *X_test*. Store the predictions in a variable named `'y_preds'`. Note: Since the model has been trained on scaled data (i.e., both features and targets, the predictions will also be in the “scaled” space. We need to transform the predictions back to the original space).
- iv. (2 points) Use [inverse_transform\(\)](#) function to convert the normalized data (`'y_preds'`) to original scale. Store the transformed values back into `'y_preds'`.
- v. (3 points) Perform [PCA](#) on the features (*X_train*) and set *n_component* as 2.
 1. Show a scatter plot where on the x-axis we plot the first PCA component and second component on the y-axis.
 2. Calculate the total percentage of variance captured by the 2 PCA components using [pca.explained_variance_ratio_](#). Also, report the strength of each PCA component using [pca.singular_values_](#).
 - i. 51.413 %
 - ii. [236.45122144 179.81942968]

d. Evaluation: (6 points)

- i. (2 points) Plot a scatter plot using [matplotlib.pyplot.scatter](#) function. Plot the predicted median house values on the y-axis vs the actual median house values on the x-axis.
- ii. (4 points) Calculate [MAPE](#), [RMSE](#) and [R²](#) for the model and report them in the following table.
Hint for RMSE set the *squared* parameter to False.

Model	MAPE	RMSE	R ²
Linear Regression	0.2845726683542122	68551.8723828916	0.6495809082844279

e. **Discussion: (8 points)**

- i. (2 points) Based on the weights of the linear regression model, rank the features (note: only continuous features) in decreasing order of influence on the predictions.
 - i. Median Income
 - ii. Total_bedroom
 - iii. Households
 - iv. House median age
 - v. Total_Rooms
 - vi. Population
 - vii. Latitude
 - viii. Longitude

- ii. (2 points) Discuss how the influence of the features (obtained in question e.1) relates to the pair-wise correlation results calculated above i.e., are the features that are highly correlated with the output also the most influential or is there some other phenomenon being observed?

- i. Median Income
- ii. Total_bedroom
- iii. Households
- iv. House median age
- v. Total_Rooms
- vi. Population

In most cases, the features with the highest correlation tend to have the greatest influence. However, an anomaly occurs when considering total bedrooms and total rooms, as their real-world relationship creates a different correlation and influence order. This is because total rooms includes total bedrooms in its count. Otherwise, the correlation and influence order is usually the same.

- iii. (2 points) Comment about the MAPE, RMSE, R^2 results. What can we learn from each of these results about the model prediction performance?

- i. MAPE (Mean Absolute Percentage Error) is a measure of prediction accuracy for a forecasting method that measures the average magnitude of the errors in percentage terms.
Mean Absolute Percentage Error (MAPE) is a commonly used metric for assessing the accuracy of a model's predictions. It measures the average magnitude of errors produced by a model, or how far off its predictions usually are. The lower the MAPE value, the closer the model predictions are to the true values.
- ii. RMSE (Root Mean Squared Error) measures the average magnitude of the error by taking the square root of the average of squared differences between forecast and actual values.
Root Mean Square Error (RMSE) or Root Mean Square Deviation (RMSD) is a commonly used measure for evaluating the accuracy of predictions. It shows the difference between the predicted values and the true values, measured by the Euclidean distance. The lower the RMSE or RMSD, the better the quality of the predictions.
- iii. R^2 (Coefficient of Determination) is a statistic that measures how well a model fits the data. It is used to determine the degree of variation in a data set that can be explained by the model.
The R^2 score is a highly significant metric used to assess the performance of a regression-based machine learning model. It is also known as the coefficient of determination. It works by measuring the extent to which the variance in the predictions is explained by the dataset. The best result is 1, indicating that the model explains all of the variance in the predictions; a score of 0 indicates that the model does not explain any relationship, and a negative score implies that the model is a supermodel.

- iv. (2 points) Why is centering and scaling the data important before performing PCA?
 - i. Prior to performing eigenanalysis, mean-centering the input data is an essential part of the process. This helps to guarantee that the first principal component is proportionate to the most significant variance of the given data.
 - ii. Scaling the data is important as the measures we used in the data were different and hence to standardize the data we scale it to data is working with same unit of measure. Since PCA tries to get the features with maximum variance, If we don't standardize the data PCA might skew towards the high magnitude features as the variance is high for high magnitude features which might lead to improper selection of principle components

Q2. Titanic Classification Problem (33 points)

- a. **Dataset Description:** The sinking of the Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the widely considered “unsinkable” RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren’t enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew. While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others. Our goal is to develop a classifier to predict whether a passenger *survived* the calamity. The attributes in the datasets are as follows:
- PassengerID (categorical): Passenger ID
 - Ticket (categorical): Ticket Number
 - Name (categorical): Passenger Name
 - Cabin (categorical): Passenger Cabin
 - Pclass (categorical): Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
 - Sex (categorical): Passenger Sex
 - Age (continuous): Passenger Age
 - SibSp (continuous): Number of Siblings/Spouses Aboard
 - Parch (continuous): Number of Parents/Children Aboard
 - Fare (continuous): Passenger Fare
 - Embarked (categorical): Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
 - Survived (categorical): Survival (0 = No; 1 = Yes) --- this is the target column. Make sure to encode data appropriately for each classifier.
 - Our target variable will be Survived. Use the rest of the fields mentioned above to predict whether a passenger survived the Titanic shipwreck.
- b. **Data Loading / Preprocessing (10 points)**
- Loading (4 points)
 - Load the data <df_train.csv> and <df_test.csv> as a pandas dataframe using the ‘pandas.read_csv’ function. The ‘df_test.csv’ has been preprocessed (I.e., null values have been dropped, certain columns etc. have been dropped) and should not be changed apart from splitting the dataframe into X_test and y_test. The ‘df_train’ data has NOT been preprocessed and you will need to preprocess and prepare the ‘df_train’ dataframe. Note: Neither *df_train* nor *df_test* have been scaled. The next few steps will enumerate data preprocessing, scaling requirements we need to perform.
 - The resulting dataframe (i.e., *df_train*) should have the shape (712,12) indicating that there are 712 instances and 12 columns.
 - In *df_train*, dataframe, currently you have 12 columns which are the following – PassengerID, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked and the Survived column (target variable).
 - Use the ‘pandas.isnull().sum()’ function check if there are any missing values in the *df_train* dataframe. Report which columns have missing (i.e., null) values and provide the number of the null values in the columns.
 - Use the ‘pandas.DataFrame.drop()’ function to drop the ‘Cabin’, ‘PassengerID’, ‘Name’ and ‘Ticket’ columns.
 - Use the ‘pandas.DataFrame.fillna()’ function to replace the NA values in the ‘Age’ column with the mean value of the ‘Age’ column. Note: This process is called ‘imputation’ (i.e., filling null values with a pre-specified value) and we are employing one strategy called *mean imputation*, but other strategies can also be employed in general. Use the ‘dropna()’ function to drop any remaining rows that consist of NA values.
 - Your task is to use the feature columns to predict the target column. This can be cast as a classification problem.

8. Create a pandas dataframe `X_train` of features (by dropping the 'Survival' column from the `df_train` dataframe). Create a pandas *Series* object of targets `y_train` (by only considering the 'Survival' column from the `df_train` dataframe). Moving forward, we will be working with `X_train` and `y_train`. At this point also split the `df_test` into `X_test` and `y_test` by dropping the 'Survival' column and storing the features into `X_test`. Store the 'Survival' column in `y_test`.

ii. Data Visualization (4 points)

1. Using matplotlib employ a scatter plot using '`matplotlib.pyplot.scatter`' between the age of the passengers and the price of their fare. Label the x-axis and the y-axis along with the giving the plot a title.
 - a. What is the highest and lowest fare price?
 - i. max fare 512.3292
 - ii. min fare 0.0
 - b. What are the respective mean values of these two features?
 - i. mean fare 32.658
 - ii. mean age 29.81
 - c. What was the age of the oldest passenger?
 - i. 80
2. **Only for this question** use the `df_train` dataframe. Using matplotlib visualize the number of males and females that survived and their respective *passenger classes* on two separate bar chart plots using '`matplotlib.pyplot.bar`' (Passenger Class column).
 - a. Which *class of passengers* had the least number of survived males and how many? Repeat this analysis for females.
 - i. For males least class is 2 with 15 survivors
 - ii. For females least class is 2 with 50 survivors
 - b. Which *class of passengers* had the greatest number of survived males and how many? Repeat this for females.
 - i. Males class 1 with 42 survivors
 - ii. Females class 1 with 74 survivors
3. Using the Target variable (Survived) in `y_train` plot a bar chart showing the distribution of the 'Survived' column.
 - a. What initial comment can you make about this distribution in terms of the frequency of each class?
 - i. In terms of `y_train` plotted as a bar chart the frequency distribution is uneven as the number of entries for Survivor = 0 are more than the entry points for Survivor = 1 with frequency for 1 being almost half of frequency for 0
 - ii. In terms of bar chart of total number of survivors for each pclass we can see that frequency of survivors is almost evenly distributed with class 1 having maximum frequency, class 2 having least and class 3 in the middle of all frequencies.
4. So far you should have successfully been able to load, preprocess and visualize your data. Use the '`pd.get_dummies()`' function to convert categorical data into dummy variables ('Sex' and 'Embarked'). **(Perform this only on `X_train` store the result back into `X_train`).**
 - a. What is the new shape of `X`?
 - i. Before it was (710, 7)
 - ii. After `get_dummies` (710, 8) using `drop_first=1`

iii. Data Scaling (2 points)

1. Employ `X_train` and `MinMaxScaler` only on the continuous attributes. Employ the '`fit_transform()`' function of the scaler to retrieve the new (scaled) version of the data. Store the scaled values in `X_train` again.
2. Scale the `X_test` using the scaler you have just fit, this time using the '`transform()`' function. Note: store the scaled values back into `X_test`. At the end of this step, you must have `X_train`, `X_test`, all scaled according to the `MinMaxScaler`.

c. Modelling (10 points)

- i. (2 points) Modelling (Model Instantiation / Training) using **Logistic Regression classifier**
 1. Employ the Logistic Regression classifier from sklearn and instantiate the model. Label this model as '`model_lr`'
 2. Once instantiated, '`fit()`' the model using the *scaled* `X_train`, `y_train` data.

3. Employ the 'predict()' function to obtain predictions on X_test and store this in a variable labeled as 'y_pred_lr'.
 4. Employ the 'accuracy_score' function by using the 'y_pred_lr' and 'y_test' variables as the functions parameters and print the accuracy of the Logistic Regression model.
- ii. (2 points) **Modelling (Model Instantiation / Training) using Support Vector Machine Classifier**
1. Employ the Support Vector Machine (SVM) classifier from sklearn () and instantiate the model. Label this model as 'model_svm'
 2. Once instantiated, 'fit()' the model using the *scaled* X_train, y_train data.
 3. Employ the 'predict()' function to obtain predictions on X_test and store this in a variable labeled as 'y_pred_svm'.
 4. Employ the 'accuracy_score' function by using the 'y_pred_svm' and 'y_test' variables as the functions parameters and print the accuracy of the SVM model.
- iii. (3 points) **Modelling Logistic Regression Classifier** with the addition of noise on the target variable. In the data repository you should see three noisy datasets – 'df_train_noise20', 'df_train_40', 'df_train_60'. These datasets have already been preprocessed. In each dataset 'df_train_noise<integer>', the integer indicates the percentage of noise injected into the target variable in that training set. The noise can be considered a result of incorrect class labelling of a particular instance. For example, in df_train_noise20, 20% of the instances have an incorrect target label in the training set. Our goal will be to train a set of classification models on such noisy training data and test on a clean test set (i.e., same as what we have been using so far 'df_test').
- Load the 'df_train_noise<nl>' (nl means noise level and is a place holder for the integer percentage) datasets and split the data into X_train_<nl> and y_train_<nl> (e.g., if working with 'df_train_noise20' we would split the data and store it in variables named X_train_20, y_train_20). 'X_train_<nl>' should store the features and 'y_train_<nl>' should store the target variable.
- iv. Repeat the following steps (1 – 4) for the 20%, 40%, 60% noise level datasets. Train a new Logistic Regression model on the new training and use the pre-existing X_test and y_test to evaluate your model. Label this model as 'model_lr_noise_<nl>'. Specifically, do the following:
1. Employ a new Logistic Regression classifier from sklearn and instantiate the model. Label this model as 'model_lr_noise_<nl>'
 2. Once instantiated, 'fit()' the model using the X_train_<nl> and y_train_<nl> data.
 3. Employ the 'predict()' function to obtain predictions on X_test and store this in a variable labeled as 'y_pred_lr_noise_<nl>'.
 4. Employ the 'accuracy_score' function and print the accuracy of the new Logistic Regression model.
 - a. What is your initial observation of the accuracy, is the accuracy higher or lower than that of the clean dataset?
 - i. Accuracy is better in clean data set compared to noisy sets
 - ii. Accuracy of clean data set = 0.888268156424581
 - iii. Accuracy of noisy data sets :
 1. 20% noise = 0.8435754189944135
 2. 40% noise = 0.7653631284916201
 3. 60% noise = 0.6089385474860335
- v. (3 points) **Modelling Support Vector Machine (SVM) Classifier** with the addition of noise on the target variable. In the data repository you should see three noisy datasets – 'df_train_noise20', 'df_train_40', 'df_train_60'. These datasets have already been preprocessed. In each dataset 'df_train_noise<integer>', the integer indicates the percentage of noise injected into the target variable in that training set. The noise can be considered a result of incorrect class labelling of a particular instance. For example, in df_train_noise20, 20% of the instances have an incorrect target label in the training set. Our goal will be to train a set of classification models on such noisy training data and test on a clean test set (i.e., same as what we have been using so far 'df_test').
- Load the 'df_train_noise<nl>' (nl means noise level and is a place holder for the integer percentage) datasets and split the data into X_train_<nl> and y_train_<nl> (e.g., if working with 'df_train_noise20' we would split the data and store it in variables named X_train_20, y_train_20). 'X_train_<nl>' should store the features and 'y_train_<nl>' should store the target variable.
- Repeat the following steps (1 – 4) for the 20%, 40%, 60% noise datasets.
- Train a new SVM classification model on the new training and use the pre-existing X_test and y_test to evaluate your model. Label this model as 'model_svm_noise_<nl>'. Specifically, do the following:
1. Employ a new SVM classifier from sklearn and instantiate the model. Label this model as 'model_svm_noise_<nl>'

2. Once instantiated, 'fit()' the model using the X_train_<nl> and y_train_<nl> data.
3. Employ the 'predict()' function to obtain predictions on X_test and store this in a variable labeled as 'y_pred_svm_noise<nl>'.
 4. Employ the 'accuracy_score' function and print the accuracy of the new Logistic Regression model.
 - a. What is your initial observation of the accuracy, is the accuracy higher or lower than that of the clean dataset?
 - i. Accuracy is better in clean data set compared to noisy sets
 - ii. Accuracy of clean data set = 0.8659217877094972
 - iii. Accuracy of noisy data sets :
 1. 20% noise = 0.8324022346368715
 2. 40% noise = 0.8547486033519553
 3. 60% noise = 0.6312849162011173

d. Evaluation (5 points)

- i. (3 points) Report F1 Score, Precision, Recall, Accuracy (All on the test set X_test, y_test)
 1. Employ a 'classification_report()' function from sklearn.metrics to report the precision recall and f1 score for each class for the 'model_lr' model and the 'model_svm' along with a confusion matrix for each of them.
- ii. (2 points) Report the accuracy and classification report for each of the three noisy models (model_lr_noise<nl>, model_svm_noise<nl>).

e. Discussion (8 points)

- i. (4 points) Compare the performance of the Logistic Regression (model_lr) classifier and the SVM classifier (model_svm) trained on the clean training data sets. Using the classification reports which model performs better? Mention the specific numbers (i.e., from your results calculating the precision, recall, F1 score) in a table.
 1. The F1 score is a measure of accuracy that indicates how well a particular class was predicted.
 2. We can see that the logistic regression model had a better f1 score than the svm model for both classes of Not Surviving and Surviving (0 and 1 respectively)

Model Name	Class	Precision	Recall	F1
model_lr	0	0.8974358974358975	0.9292035398230089	0.9130434782608695
model_lr	1	0.8709677419354839	0.8181818181818182	0.84375
model_svm	0	0.8803418803418803	0.911504424778761	0.8956521739130435
model_svm	1	0.8387096774193549	0.7878787878787878	0.8125

- ii. (4 points) Report the performance of all the classification models you have trained thus far in terms of (precision, recall, F1 scores). From the results of the three noisy models implemented using Logistic Regression. What did you notice about the accuracy as the noise of the dataset increased? What can you say about the effect of noise on data mining pipelines?
 1. The f1 score demonstrates that an increase in noise in the training data is reflective of a decrease in accuracy. This is a major problem within data mining and there is a need for identifying and avoiding datasets with high levels of noise. Without proper identification, the accuracy of the data mining can drastically drop.

Model Name	Class	Precision	Recall	F1
model_lr	Not Survived	0.8974358974358975	0.9292035398230089	0.9130434782608695
model_lr	Survived	0.8709677419354839	0.8181818181818182	0.84375
model_lr_noise20	Not Survived	0.8455284552845529	0.9203539823008849	0.8813559322033898
model_lr_noise20	Survived	0.8392857142857143	0.8392857142857143	0.7704918032786886
model_lr_noise40	Not Survived	0.738255033557047	0.9734513274336283	0.8396946564885496
model_lr_noise40	Survived	0.9	0.4090909090909091	0.5625000000000001
model_lr_noise60	Not Survived	0.6776859504132231	0.7256637168141593	0.7008547008547009
model_lr_noise60	Survived	0.46551724137931033	0.4090909090909091	0.43548387096774194
model_svm	Not Survived	0.8803418803418803	0.911504424778761	0.8956521739130435
model_svm	Survived	0.8387096774193549	0.7878787878787878	0.8125
model_svm_noise20	Not Survived	0.7902097902097902	1.0	0.8828125
model_svm_noise20	Survived	1.0	0.5454545454545454	0.7058823529411764
model_svm_noise40	Not Survived	0.8222222222222222	0.9823008849557522	0.8951612903225806
model_svm_noise40	Survived	0.9545454545454546	0.6363636363636364	0.7636363636363637
model_svm_noise60	Not Survived	0.6715328467153284	0.8141592920353983	0.736
model_svm_noise60	Survived	0.5	0.3181818181818182	0.3888888888888889

Q3. Bank Churn Classification Problem (33 points)

- a. **Dataset Description:** Banking is one of those traditional industries that has gone through a steady transformation over the past few decades. Yet, many banks today with a sizeable customer base are hoping to gain a competitive edge but have not tapped into the vast amounts of data they have, especially in solving one of the most acknowledged problems – *customer churn* (i.e., a customer leaving the bank). It is advantageous to banks to know what leads a client to leave the bank. Banks often use the customer churn rate as one of their key business metrics because the cost of retaining existing customers is far less than acquiring new ones, and meanwhile increasing customer retention can greatly increase profits.

Churn prevention allows companies to develop different programs such as loyalty and retention programs to keep as many customers as possible. Following are the attributes of the dataset we will be working with.

- i. RowNumber (continuous) — corresponds to the record (row) number and has no effect on the output.
- ii. CustomerId (categorical)— contains random values and has no effect on customer leaving the bank.
- iii. Surname (categorical)— the surname of a customer has no impact on their decision to leave the bank
- iv. CreditScore (continuous) — can influence customer churn, since a customer with a higher credit score is less likely to leave the bank.
- v. Geography (categorical) — a customer's location can affect their decision to leave the bank.
- vi. Gender (categorical) — it's interesting to explore whether gender plays a role in a customer leaving the bank.
- vii. Age (continuous) — this is certainly relevant, since older customers are less likely to leave their bank than younger ones.
- viii. Tenure (continuous) — refers to the number of years that the customer has been a client of the bank. Normally, older clients are more loyal and less likely to leave a bank.
- ix. Balance (continuous) — also a very good indicator of customer churn, as people with a higher balance in their accounts are less likely to leave the bank compared to those with lower balances.
- x. NumOfProducts (continuous) — refers to the number of products that a customer has purchased through the bank.
- xi. HasCrCard (categorical) — denotes whether a customer has a credit card. This column is also relevant since people with a credit card are less likely to leave the bank.
- xii. IsActiveMember (categorical) — active customers are less likely to leave the bank.
- xiii. EstimatedSalary (continuous) — as with balance, people with lower salaries are more likely to leave the bank compared to those with higher salaries.
- xiv. Exited (Categorical) — whether or not the customer left the bank. (**Target variable**)

b. Data Loading / Preprocessing (10 points)

- i. Loading (2 points)
 1. Load the data <BankChurn.csv> as a pandas dataframe using the `pd.read_csv()` function which returns a dataframe, store this value in a variable named 'df'.
 2. The resulting dataframe should have the shape (10000,14) indicating that there are 10000 instances and 14 columns.
 3. In this dataframe, currently you have 9 features which are the following: RowNumber, CustomerID, Surname, CreditScore, Geography, Gender, Age, Tenure, Balance, NumOfProducts, HasCrCard, IsActiveMember, EstimatedSalary. Using the `'pandas.dataframe.drop'` function to drop the RowNumber, CustomerID and Surname columns.
 4. Using the `'pandas.isnull()'` function check if there are any missing values in the dataframe and report this value (i.e., the number of missing values per column of the dataframe).

5. Your task is to use feature columns to predict the target column (which is categorical in our case). This can be cast as a classification problem.
6. Create a dataframe X of features (by dropping the 'Exited' column from the original dataframe). Create a Pandas Series object of targets Y (by only considering the 'Exited' column from the original dataframe). Moving forward, we will be working with X and Y.

ii. Data Visualization (4 points)

1. Visualize the distribution of the 'Age' and 'CreditScore' column using the 'matplotlib.pyplot.hist' function as two separate plots. Label the x-axis and the y-axis along with giving the plot a title and assign a bin size of 7.
 - a. What are the respective mean values of these two features (use the 'pandas.DataFrame.mean()' function)?
 - i. mean of Age is 38.9218
 - ii. mean of CreditScore is 650.5288
 - b. What is the respective standard deviation of these two features (use the 'pandas.DataFrame.std()' function)?
 - i. std of Age is 10.487806451704609
 - ii. std of CreditScore is 96.65329873613035
2. **Only for this question** use the dataframe consisting of the target variable (initialized as 'df'). Using matplotlib visualize the number of males and females in each country who are active members and not active members. (Visualize this using a **barchart**. You will need to use the 'Gender', 'Geography' and 'IsActiveMember' features for this question). Visualize these graphs on two separate plots with respect to their active status. To create a barchart using matplotlib use the 'matplotlib.pyplot.bar()' function. Also label the x-axis, y-axis and give the plots a title.
 - a. How many males are from France and are active members?
 - i. Male France active 1429
 - b. How many females are from Spain and are active members?
 - i. Female Spain active 563
 - c. How many males are from France or Germany who are not active members?
 - i. Male France inactive 1324
 - ii. Male Germany inactive 627
 - iii. Total 1,951
3. Using the target variable in Y plot a bar chart showing the distribution of the 'Exited' column (To create a barchart using matplotlib use the 'matplotlib.pyplot.bar()' function).
 - a. What can be said about this distribution (specifically keeping in mind this distribution represents the target variable) will this have an impact on the results of the classification model?
 - i. The frequency of data entries with exited as 1 / True is significantly larger than the frequency of data entries with exited as 0.
 - ii. Hence can observe that most users did not exit, leading to an unbalanced dataset. This could cause our model to be skewed towards one outcome, as the prediction will be based on a majority of the data.
4. So far you should have successfully been able to load, preprocess and visualize your data. Now, use the 'pd.get_dummies()' function to convert categorical data into dummy variables ('Gender' and 'Geography'). **(Perform this only on X).**
 - a. What is the shape of X?
 - i. Using X=pd.get_dummies(X,columns=['Gender','Geography'],drop_first=True) since there is no mention of whether we need to drop the unnecessary columns using drop_first = True we get 11 columns
 - ii. Before (10000, 10)
 - iii. After get dummies (10000, 11)

iii. Data Splitting (1 point)

1. Split data into training and test sets using the sklearn '[train_test_split\(\)](#)' function in a **80:20** ratio. The result of your data split should be X_train, X_test, y_train, y_test. (Respectively your training features, testing features, training targets and testing target arrays).

iv. Data Scaling (3 points)

1. Employ the '[MinMaxScaler](#)' function on the continuous attributes in X_train. Employ the '[fit_transform\(\)](#)' function of the scaler to retrieve the new (scaled) version of the training data (i.e., [fit_transform\(\)](#) should be run on 'X_train'). Store the result in X_train again.
2. Scale the X_test data using the scaler you have just *fit*, this time using the '[transform\(\)](#)' function. Note: store the scaled values back into X_test. At the end of this step, you must have X_train, X_test, scaled according to the MinMaxScaler.

c. Modelling (10 points)

i. (2 points) Modeling (Model Instantiation / Training) using Logistic Regression classifier

1. Employ the [Logistic Regression classifier](#) from sklearn and instantiate the model. Label this model as 'model_1_lr'
2. Once instantiated, '[fit\(\)](#)' the model using the *scaled* X_train, y_train data.
3. Employ the '[predict\(\)](#)' function to obtain predictions on X_test and store this in a variable labeled as 'y_pred_lr'.
4. Employ the '[accuracy_score\(\)](#)' function by using the 'y_pred_lr' and 'y_test' variables as the functions parameters and print the accuracy of the Logistic Regression model

ii. (2 points) Modeling (Model Instantiation / Training) using Support Vector Machine Classifier

1. Employ the Support Vector Machine (SVM) classifier from sklearn and instantiate the model. Label this model as 'model_2_svm'
2. Once instantiated, '[fit\(\)](#)' the model using the *scaled* X_train, y_train data.
3. Employ the '[predict\(\)](#)' function to obtain predictions on X_test and store this in a variable labeled as 'y_pred_svm'.
4. Employ the '[accuracy_score](#)' function ('[sklearn.metrics.accuracy\(\)](#)' function) by using the 'y_pred_lr' and 'y_test' variables as the functions parameters and print the accuracy of the SVM model.

iii. (2 points) Modeling Logistic Regression Classifier on a *balanced* dataset

1. Employ Synthetic Minority Oversampling on X_train and y_train. To use SMOTE you will have to install the imbalanced-learn library, this can either be down by executing the following command '***pip install -U imbalanced-learn***' command '***conda install -c conda-forge imbalanced-learn***' command for the Anaconda Cloud platform. (For more information click the following link: <https://imbalanced-learn.org/stable/install.html>). Import the 'SMOTE' function from the '[imblearn.over_sampling](#)'. Use the '[smote.refit_resample\(\)](#)' function on X_train and y_train using its default parameters. Store them in X_train_smote, y_train_smote. - Be careful to employ SMOTE ONLY on the training data and not on the full dataset because that can cause inadvertent "data leakage" (please see: <https://arxiv.org/pdf/2107.00079.pdf> for details) .
2. Employ a new Logistic Regression classifier from *sklearn* and instantiate the model. Label this model as 'model_3_smote_lr'
3. Once instantiated, '[fit\(\)](#)' the model using the *balanced* X_train_smote, y_train_smote data.
4. Employ the '[predict\(\)](#)' function to obtain predictions on X_test and store this in a variable labeled as 'y_pred_smote_lr'.

5. Employ the 'accuracy_score' function by using the 'y_pred_lr' and 'y_test' variables as the functions parameters and print the accuracy of the new Logistic Regression model.
 - a. What is your initial observation of the accuracy of model_3_smote_lr vs. accuracy of model_1_lr? What could be the reasoning for (any possible) change in accuracy?
 - i. Accuracy of model_3_smote_lr = 0.704
 - ii. accuracy of model_1_lr = 0.8125
 - iii. Recall of model_3_smote_lr [class 0 , class 1] = [0.70 , 0.71]
 - iv. Recall of model_1_lr [class 0 , class 1] = [0.96 , 0.22]
 - v. Recall in machine learning is a measure of how well a model predicts true positive results. It measures the proportion of positive results that are correctly identified as such by the model. It is also referred to as sensitivity or the true positive rate.
 - vi. We had a poor recall for class 1 , but a higher recall for the class 0. To improve the model, we opted to over-sample, which traded the class 0's recall for increased recall on class 1 data. This led to a decline in accuracy, but created a much better model, as the previous model was not performing adequately for either class, making performance somewhat equal for both.
- iv. **(2 points) Modeling SVM on a balanced dataset**
 1. Employ Synthetic Minority Oversampling on X_train and y_train. Import the 'SMOTE' function from the 'imblearn.over_sampling'. Use the 'smote.refit_resample()' function on X_train and y_train. Store them in X_train_smote, y_train_smote.
 - a. At the end of this step, your new training set i.e., (X_train_smote , y_train_smote) should have the same number of instances for each of the two classes.
 2. Employ a new SVM classifier from *sklearn* and instantiate the model. Label this model as 'model_4_smote_svm'
 3. Once instantiated, 'fit()' the model using the *balanced* X_train_smote, y_train_smote data.
 4. Employ the 'predict()' function to obtain predictions on X_test and store this in a variable labeled as 'y_pred_smote_svm'.
 5. Employ the 'accuracy_score' function ('sklearn.metrics.accuracy()' function) by using the 'y_pred_lr' and 'y_test' variables as the functions parameters and print the accuracy of the new SVM model.
 - a. What is your initial observation of the accuracy of model_4_smote_svm vs. accuracy of model_2_svm? What could be the reasoning for (any possible) change in accuracy?
 - i. accuracy of model_4_smote_svm 0.7135
 - ii. accuracy of model_2_svm 0.7975
- v. **(2 points) Modeling Grid Search Parameter Selection for SVM**
 1. We will now be reverting to our X_train and y_train data. Initialize a variable labeled as 'param_grid' storing the following: {'gamma': [0.001, 0.01, 0.1], "C": [1,10,100,1000,10000]}.
 2. Employ the gridsearchCV function and initialize the following parameters: estimator = SVC(), param_grid = param_grid, cv=5, verbose =1, scoring = 'accuracy'
 3. Once instantiated, 'fit()' the model using the X_train_smote, y_train_smote data.
 4. Print the best paramaters using the 'best_params_' attribute and print the mean cross validated score of the best estimator (hint use the 'best_score_' attribute).
 5. Employ the 'score' function by using the 'X_test' and 'y_test' variables as the functions parameters and print the accuracy of the new gridsearch SVM model.SVM model.

d. **Evaluation (5 points)**

- i. (2 points) Calculate F1 Score, Precision, Recall, Accuracy (All on the test set X_{test} , y_{test})
 1. Employ the ``classification_report()`` function from `sklearn.metrics` to report the precision recall, f1 score and accuracy for each class for the first **four** models (**parts c.i – c.iv**).
- ii. (2 points) Visualize a confusion matrix for the first four models
 1. Employ the ``confusion_matrix()`` function from `sklearn.metrics` to report the confusion matrix results.
 2. Report the False Negative and False Positive values for `model_1_lr`.
 - a. False Negative = 63
 - b. False Positive = 312
- iii. (1 point) Report the best F1 score of the grid search implemented in the fifth model (**part c.v**). Also report the best parameters from the grid search on the training set.
 1. Best params {'C': 1000, 'gamma': 0.1}
 2. Best f1 score 0.854125

e. **Discussion (8 points)**

- i. (2 points) Compare the performance of the Logistic Regression (`model_1_lr`) classifier and the SVM classifier (`model_2_svm`). Using the classification reports which model performs better? Mention the specific numbers (i.e., from your results calculating the precision, recall, F1 score) in a table.

Model Name	Class	Precision	Recall	F1
model_1_lr	0	0.8308026030368764	0.9605015673981191	0.8909566734515847
model_1_lr	1	0.5961538461538461	0.22962962962962963	0.3315508021390374
model_2_svm	0	0.8534342888047594	0.9893416927899686	0.9163763066202091
model_2_svm	1	0.8874172185430463	0.3308641975308642	0.48201438848920863

- ii. (2 points) Compare the results between the models prior to SMOTE and after. Use a combination of the classification report and the confusion matrix to emphasize your observation. (`model_1_lr` vs `model_3_lr` and `model_2_svm` vs `model_4_svm`)
 1. After applying SMOTE, the F1 score for class 0 decreases while the F1 score for class 1 increases. As we can see from the confusion matrix in the jupyter notebook, although the precision decreases slightly due to an increase in false positives, the recall increases significantly due to the reduction of false negatives. This increase in recall is crucial for obtaining meaningful predictions.

Model Name	Class	Precision	Recall	F1
model_1_smote_lr	0	0.9057971014492754	0.7053291536050157	0.7930912936200212
model_1_smote_lr	1	0.37994722955145116	0.7111111111111111	0.4952708512467756
model_2_smote_svm	0	0.9257575757575758	0.7661442006269592	0.8384219554030875
model_2_smote_svm	1	0.4514705882352941	0.7580246913580246	0.5658986175115208

- iii. (2 points) Discuss the advantages and disadvantages between oversampling and under-sampling. Use the following article to help you formulate your answer and reasoning. (<https://www.mastersindatascience.org/learning/statistics-data-science/undersampling/>)
 1. When we have less data and our data is imbalanced, we can use oversampling to create artificial data. However, this can degrade the quality of the sample and lead to poorer accuracy compared to using only natural data. One advantage of oversampling is that we need less data to train our model.
 2. On the other hand, undersampling can be used when we have a large amount of data and our data is imbalanced. The disadvantage of this approach is that we need a lot of data, but the advantage is that it does not degrade the quality of our model training because all of the data is natural. With undersampling, we include all rare cases but remove unnecessary cases that do not contribute to the model.
- iv. (2 points) From the results of the grid-search, comment on importance of the gamma parameter on SVM performance. What effect will the gamma value have if it is too big or too small?
 1. Grid search allows us to test all possible combinations of hyperparameters and find the best one for our model. The gamma parameter determines the extent to which a single sample affects the training. When multiplied by the learning rate, gamma dampens the impact of outliers on our training.
 2. If gamma has a very small value, it will take a long time to train the model and may not even reach the optimum. On the other hand, if gamma is too large, we may overshoot the optimum and simply oscillate around it without converging.