Machine Learning Assignment 1 Theoretical Part

Name: Jayveersinh Raj

Email: j.raj@innopolis.university

Group: BS20-DataScience-01

2.1 Regarding the Preprocessing

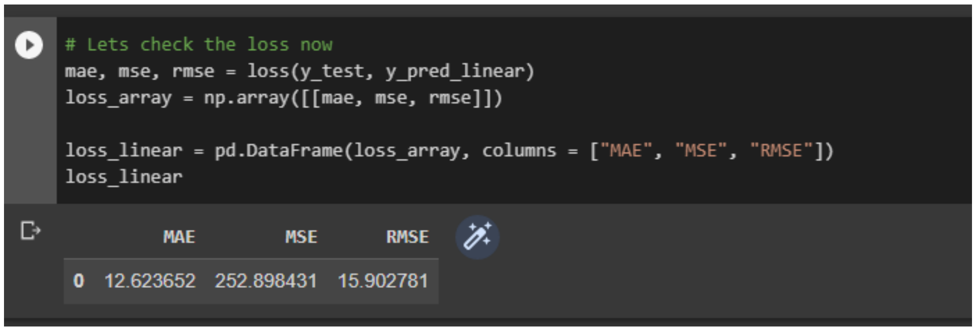
- **Which regression model was the most effective for the missing values, and why?**

Answer:

➔ ***Polynomial Regression with degree 3*** was the most effective for the missing values.

Below I provide the code and the result snippets with comparisons between them.

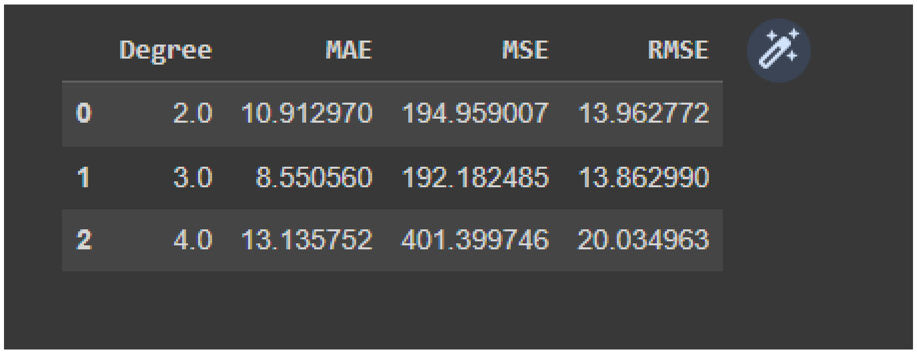
1. ***Linear Regression*** is given below



```
# Lets check the loss now
mae, mse, rmse = loss(y_test, y_pred_linear)
loss_array = np.array([[mae, mse, rmse]])

loss_linear = pd.DataFrame(loss_array, columns = ["MAE", "MSE", "RMSE"])
loss_linear
```

```
         MAE         MSE       RMSE
0  12.623652  252.898431  15.902781
```

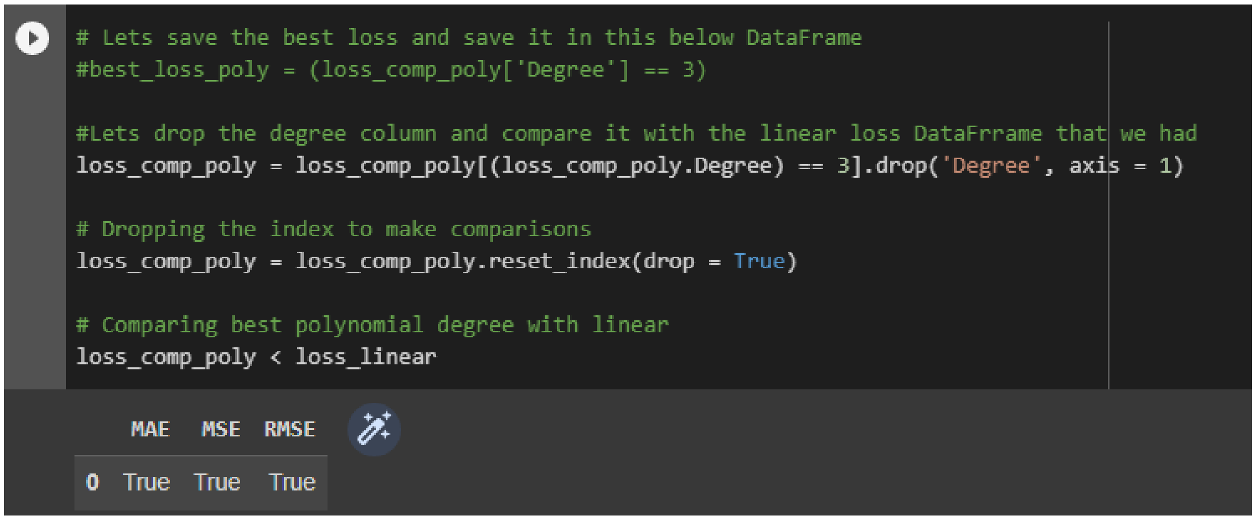
2. ***Polynomial regression*** with different powers



	Degree	MAE	MSE	RMSE
0	2.0	10.912970	194.959007	13.962772
1	3.0	8.550560	192.182485	13.862990
2	4.0	13.135752	401.399746	20.034963

➔ ***Degree 3 was found better***. In code it is described, and showed in DataFrame

➔ ***Degree 3 was even better than linear***



```
# Lets save the best loss and save it in this below DataFrame
#best_loss_poly = (loss_comp_poly['Degree'] == 3)

#Lets drop the degree column and compare it with the linear loss DataFrrame that we had
loss_comp_poly = loss_comp_poly[(loss_comp_poly.Degree) == 3].drop('Degree', axis = 1)

# Dropping the index to make comparisons
loss_comp_poly = loss_comp_poly.reset_index(drop = True)

# Comparing best polynomial degree with linear
loss_comp_poly < loss_linear
```

	MAE	MSE	RMSE
0	True	True	True

➔ **REASON:**

Degree 3 captures relationship between features well, atleast better than linear. But, as we increase the degree of the polynomial it starts to overfit. Performs well on training, but is poor with test set.

What happens is that as the degree increases, the model might contain more turning points. ***A polynomial of degree 2*** has ***one turning point***, a polynomial of ***degree 3*** has ***two turning points***, and so on. We are providing the model ***greater latitude*** to match closer to the training dataset with ***each additional turning point***. The model is most likely only catching noise at higher degrees. The real underlying trend is unlikely to be that difficult.

When ***degree is 3***, the ***test MSE*** is at its ***lowest*** and ***subsequently begins to rise***. This signifies that the ***model's performance*** on the ***test set*** is ***deteriorating***. In other words, as the ***degree increases***, the model becomes ***increasingly overfitted***.

- **What encoding technique did you use for encoding the categorical features, and why?**

Answer:

I used ***Ordinal Encoding***. ***Var3*** had name of the countries, and there were more than 100s of them (unique). It makes it very difficult to visualise it, and later trying PCA with components starting from 1 till length of column - 1 for Training and testing, if I use ***One-Hot encoding.*** The huge number of columns if one hot encoding is used, also creates huge dimensionality when using Polynomial regression.

Var6 contains yes and no values, so ***Oridnal Encoding*** is better as it assigns them either 0 or 1. ***One-Hot encoding*** would increase dimensionality here too.

Moreover, neither ***var3*** or ***var6*** is our target variable, so it might not create a problem of assuming ordinal relationship between different categorical values, as it does not matter much for our non target variables.

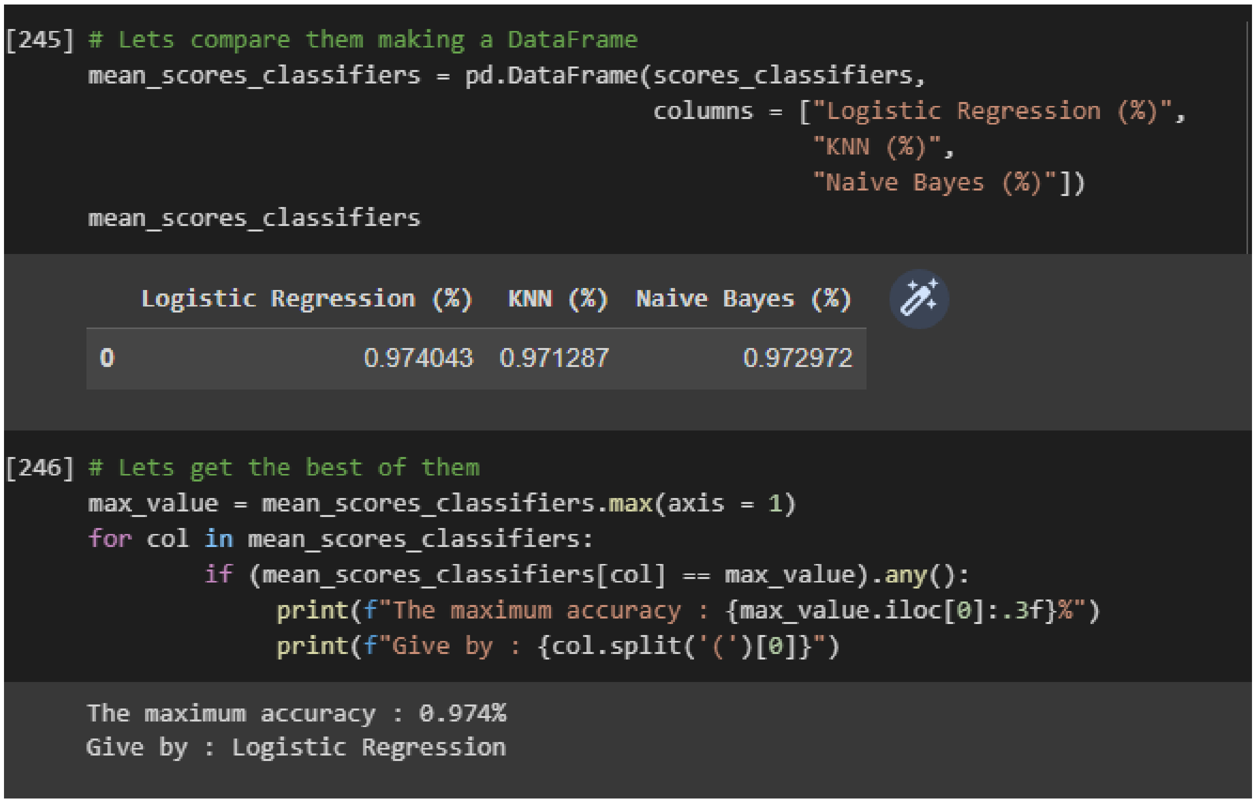
2.2 Regarding the training process

- **Which classification model performed best, and why?**

Answer:

➔ ***Logistic Regression*** performs the best out of all. Although others are close too, but ***Logistic Regression*** happens to be slightly the best.

The picture of code snippet and result with comparison is provided ahead.



```
[245] # Lets compare them making a DataFrame
      mean_scores_classifiers = pd.DataFrame(scores_classifiers,
                                             columns = ["Logistic Regression (%)",
                                                        "KNN (%)",
                                                        "Naive Bayes (%)"])
      mean_scores_classifiers
```

	Logistic Regression (%)	KNN (%)	Naive Bayes (%)
0	0.974043	0.971287	0.972972

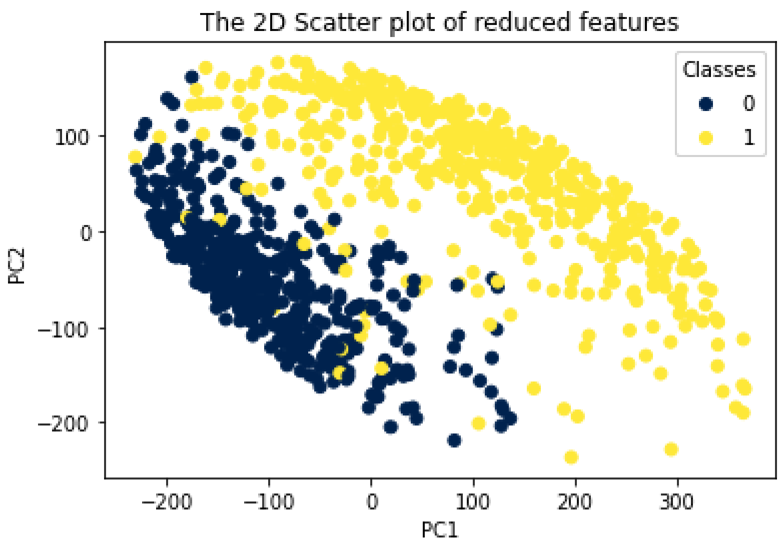
```
[246] # Lets get the best of them
      max_value = mean_scores_classifiers.max(axis = 1)
      for col in mean_scores_classifiers:
              if (mean_scores_classifiers[col] == max_value).any():
                   print(f"The maximum accuracy : {max_value.iloc[0]:.3f}%")
                   print(f"Give by : {col.split('(')[0]}")
```

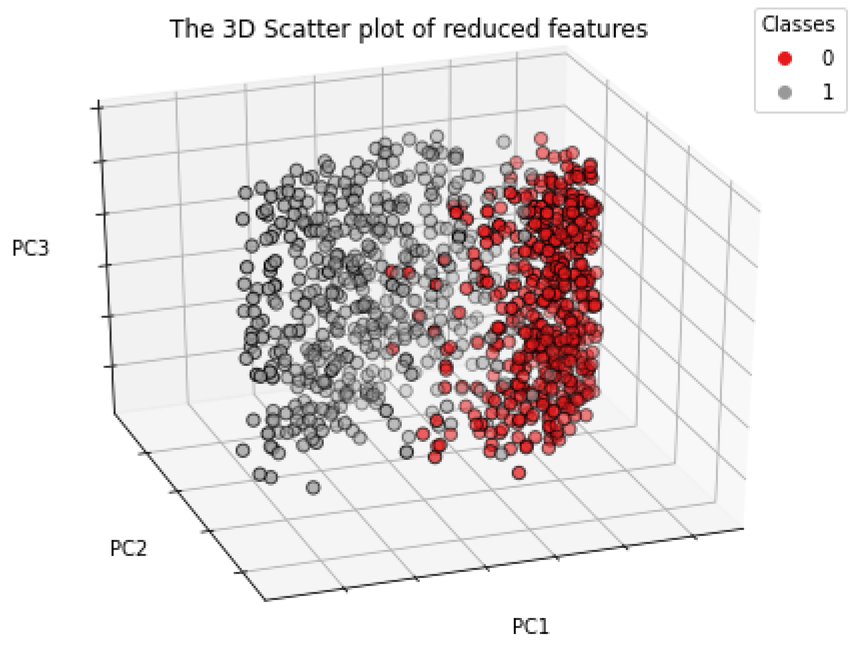
```
The maximum accuracy : 0.974%
Give by : Logistic Regression
```

➔ ***Reason:***

Logistic Regression performs well when the dataset is ***linearly separable*** and has a high accuracy for many basic data sets.

This more or less linear separablity can be seen in the pictures below, which provided plot by PCA:





- **What were the most critical features with regards to the classification, and why?**

  Answer:

  var1, var2, var4, var5 were important features. I used ***Lasso Regression*** to find them.

  Lasso regression will automatically choose the relevant characteristics while rejecting the worthless or redundant ones. In Lasso regression, removing a feature causes its coefficient to equal zero.

  So, the principle behind using ***Lasso regression*** for feature selection is straightforward: we run a Lasso regression on a scaled version of our dataset and evaluate **only features with coefficients greater than 0.** To get the desired type of Lasso regression, we must first modify the hyperparameter. Coefficients less than 0 implies that that feature is more or less redundant, and does not Effect the target drastically.

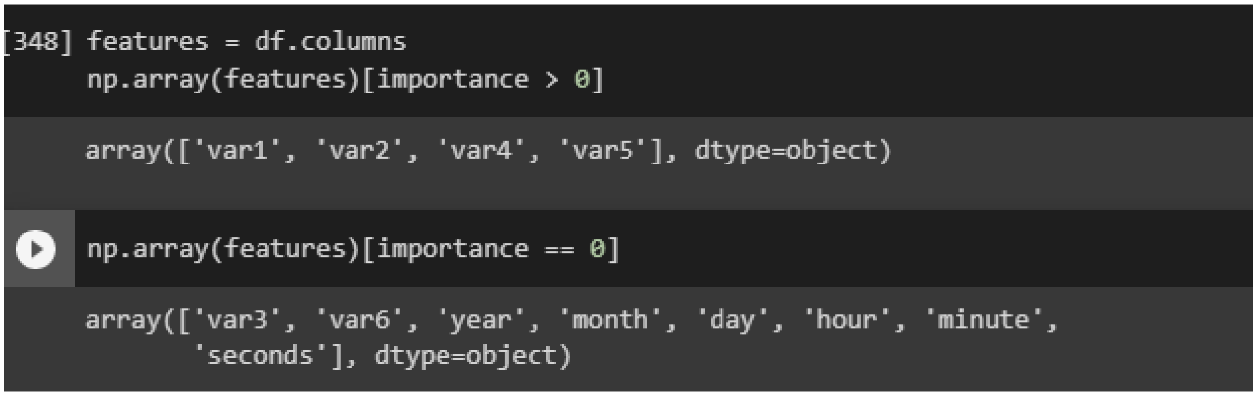
  var1, var2, var4, var5 have coefficients greater than 0, and hence are crucial.

- **What features might be redundant or are not useful, and why?**

  Answer:

  var3, var6, year, month, day, hour, minutes, seconds are redundant or useless features. In my case, I handled date time effectively, hence there exists these (***month, day, hour, minutes, seconds)*** five features.

  Reason :
  Less than 0 value of coefficients in ***Lasso Regression.*** How ***Lasso Regression*** works is explained in a nutshell in the previous question.

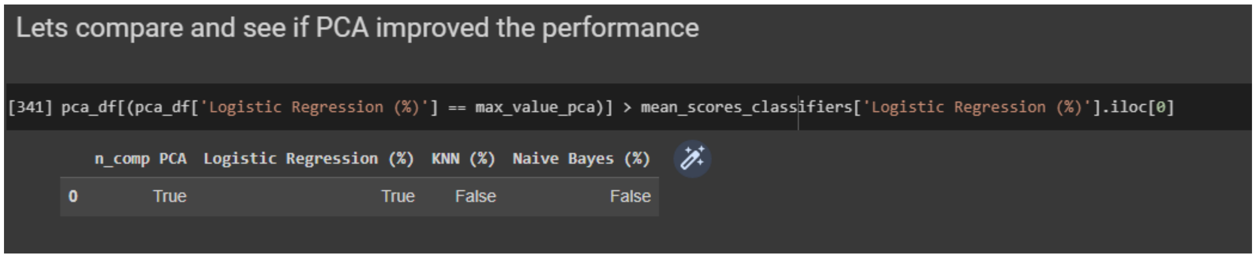


```
[348] features = df.columns
      np.array(features)[importance > 0]

      array(['var1', 'var2', 'var4', 'var5'], dtype=object)

      np.array(features)[importance == 0]

      array(['var3', 'var6', 'year', 'month', 'day', 'hour', 'minute',
             'seconds'], dtype=object)
```

- **Did the dimensionality reduction by the PCA improve the model performance, and why?**

  Answer:

  Yes, ***Principal Component Analysis*** can assist increase the performance of a Classifier. As you can see in the snippet, and result below:



Lets compare and see if PCA improved the performance

```
[341] pca_df[(pca_df['Logistic Regression (%)'] == max_value_pca)] > mean_scores_classifiers['Logistic Regression (%)'].iloc[0]
```

	n_comp PCA	Logistic Regression (%)	KNN (%)	Naive Bayes (%)
0	True	True	False	False

As we can see Logistic Regression did showed improvement on PCA.

Dimensionality reduction reduces data noise — Dimensionality reduction reduces data noise by preserving just the most significant characteristics and deleting the unnecessary features. **This boosts the model's accuracy.**

Additional Research:

A. **What is a multi-label learning problem?**

Answer:

Multi-label classification, also known as ***multi-output classification***, is a ***classification problem*** variation in which each instance may be assigned numerous nonexclusive labels. ***Multi-label classification*** is an extension of ***multiclass classification***, which is the single-label issue of classifying cases into one or more (than two) classes. The labels in the multi-label issue are nonexclusive, and there is no limit to how many classes the instance can be allocated to.

B. **Suggest an example in which you can transform the given problem into a multi-label problem? Will the models work as it is in that case, or would some changes be required?**

Answer:

➔ **An example to transform this problem into multi-label problem:**

It can be seen that ***var3*** contains ***236 country names***. If we set them as target then this problem becomes ***multiclass or multi-label problem.***

1. **Logistic Regression** : This model will not work, it would require modifications. By default it utilises ***Sigmoid function*** which is only helpful for binary classifications, as it outputs binary values. To make it useful for multi-label problems, a modified version of ***Sigmoid function, Softmax function*** is used. `multi_class='multinomial'` should be passed in the ***Logistic Regression*** function to make it work for multilabel.

2. **KNN** : KNN can be used for multi-class classifications without modifications. KNN has this advantage over others.

3. **Naive Bayes:** Works for both binary and multi-class classification. It works on the assumption of independence, and probabilities. It would work for mutli-class without modifications.

Below is a description of few important things in my notebook

---

- **How I handled data time:**

  1. I converted the ***object*** datatype of the ***var7 (Date-Time)*** series in the DataFraem to ***string*** datatype because I observed the ***whitespace*** between date and time.
  2. Then I ***split the Date-Time*** on that white space.
  3. Then from ***date*** by converting it to ***datetime*** dtype I create 3 more features, namely ***year, month, and day***
  4. Similarly I create 3 more features from ***time*** namely ***hour, minutes, seconds***
  5. Then I drop the previously existing ***var7***

- **The general design of the code:**
  I have my code fairly functional and robust. For ***classifiers*** with ***KFold k = 3,*** it is robust. ***To change k please consider changing The value of k inside the function.***

  ```
  kf = KFold(n_splits = 3).
  ```

  Change value of k from the above line in the code (k is the `n_splits`). I did not need to change the value for them all so I did not pass it as functional argument.

- **Usage / inspiration of KNN plot from labs:**
  I inspired the code for KNN plots of **Ks** from the labs. The name of variables were given the same for some the things inside the function. It was intensional because those names signifies the functionality better, so I refrained from changing them. For example naming the ***List of ks as ks.***

- **How prediction of NaN values and saving them was approached:**
    1. Saved the rows with ***NaN value in var4*** in a ***prediction set.*** Removed the ***var4*** column from that ***prediction set.***
    2. Created a ***Train and Test set.*** From the remaining ***325*** rows. I used ***300 as Training and rest 25 as test.***
    3. Then using the above ***Train and test*** datasets, I found ***degree 3 polynomial regression*** to be best performing based on comparisons with others.
    4. Then I used ***degree 3 polynomial regression*** to find the 600 missing values in ***var4*** of the ***prediction set.***
    5. Then I added the ***var4*** column to the ***prediction set.***
    6. Rearranged the columns in ***prediction set*** to match the columns with original dataset, then appended the ***prediction set*** to the ***original set.***

- **Others**

    Other than this, most of the things like ***working with pandas*** is Straight forward. ***I have added explained text cells, and even comments in the code.***

The END Thank you