Name: Jay Dinesh Mehta
UID: 2018130024
Roll No: 27
CEEL 82B, Data Science, 2022

Lab 1: Study of Linear Regression

Objective: To carry out linear regression (including multiple regression) and build a regression model using Python Platform

Case studies to consider:
1) Estimating horse fatalities from colic- use logistic regression to try to predict if a horse with colic will live or die
2) Credit Score prediction – a Model to predict the probabilities of default. Use Linear Regression to predict the probabilities of default and assign credit to potential borrowers

Outcomes:
1. To learn how to define, fit, and use a model in Python
2. To interpret the results

System Requirements: Linux/MaC/Windows OS with Anconda platform with Pandas, numpy, scipy, matplotlib, seaborn and scikit-learn ML library.

Part-A: Simple linear regression and Multiclass linear regression with data preprocessing (Handling NA values)
Use the case study relevant csv and files to build the models and evaluate the models.
General Steps:
1. Load the dataset (Use pandas)
2. Data Preprocessing (Handling NA values)
3. Exploratory Data Analysis (understanding the relationships between the variables with help of plot, scatter-plot, energy-plot etc) Use matplotlib
4. Data Partition (80% for training and 20% for testing) (Use scikit-learn)
5. Build the model (use scikit learn)
6. Summarize the model.
7. Prediction
8. Evaluate the model
9. Tuning the model

Code:

1. Load the dataset (Use pandas) and libraries

```
# Importing the libraries
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import numpy as np
import pandas as pd
import seaborn as sns
import warnings
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
```

```
[65] # Setting the color attributes
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
# Loading the dataset
data = pd.read_csv("./horse.csv", sep=',', nrows=299)
```

2. Data Preprocessing (Handling NA values)

```
data = data.drop(columns=['hospital_number', 'nasogastric_reflux_ph','abdomo_appearance','abdomo_protein'])

print('Old Size: %d' % len(data))
data = data.dropna(how = 'any', axis = 'rows')
print('New Size: %d' % len(data))

print('Check that there are no empty values after cleaning:')
is_null = pd.isnull(data).sum()
print(is_null)
data.corr()

data2 = pd.get_dummies(data, columns =['surgery','age','capillary_refill_time','surgical_lesion', 'cp_data','abdominal_distention','temp_of_extremities', 'peripheral_
data2.head()
data2 = data2.replace({'outcome': {'lived': 1, 'died': 0, 'euthanized': np.nan}})
data2 = data2.dropna(how = 'any', axis = 'rows')
```

```
Old Size: 299
New Size: 52
Check that there are no empty values after cleaning:
surgery                  0
age                      0
rectal_temp              0
pulse                    0
respiratory_rate         0
temp_of_extremities      0
peripheral_pulse         0
mucous_membrane          0
capillary_refill_time    0
pain                     0
peristalsis              0
abdominal_distention     0
nasogastric_tube         0
nasogastric_reflux       0
```

3. Exploratory Data Analysis (understanding the relationships between the variables with help of plot, scatter-plot, energy-plot etc) Use matplotlib

```python
# Handling null values (Data Preprocessing)
# Check if null values are present
is_null = pd.isnull(data).sum()
print('Empty values num:')
print(is_null)
```

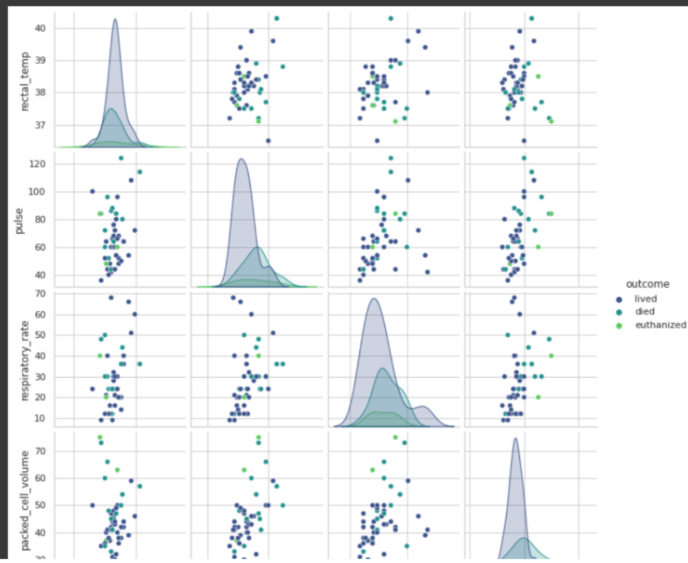
```
Empty values num:
surgery                  0
age                      0
rectal_temp              0
pulse                    0
respiratory_rate         0
temp_of_extremities      0
peripheral_pulse         0
mucous_membrane          0
capillary_refill_time    0
pain                     0
peristalsis              0
abdominal_distention     0
nasogastric_tube         0
nasogastric_reflux       0
rectal_exam_feces        0
abdomen                  0
packed_cell_volume       0
total_protein            0
outcome                  0
surgical_lesion          0
lesion_1                 0
lesion_2                 0
lesion_3                 0
cp_data                  0
dtype: int64
```

4. Data Partition (80% for training and 20% for testing) (Use scikit-learn)

```python
# Plot correlation to select best features
plt.figure(figsize=(24,12))
sns.heatmap(data.corr(),cmap='magma_r',annot=True)
```

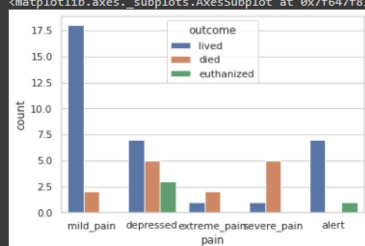
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f647fac6190>
```

```python
# Exploratory data analysis (Select features and plot them on a series)
cols = ['rectal_temp', 'pulse', 'respiratory_rate', 'packed_cell_volume', 'outcome']
sns.pairplot(data[cols], hue='outcome', palette = 'viridis');
```



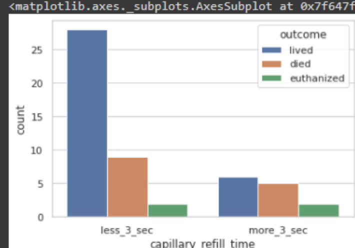

```python
# Plot of outcome vs pain levels
sns.countplot(data=data, x='pain', hue = 'outcome')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f647f81fd90>
```



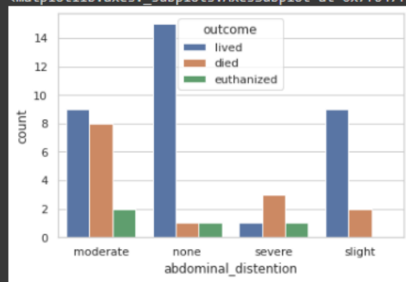

```python
[71] # Plot of outcome vs capillary_refill_time
sns.countplot(data=data, x='capillary_refill_time', hue = 'outcome')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f647f7a6290>
```



```
# Plot of outcome vs abdominal_distention
sns.countplot(data=data, x='abdominal_distention', hue = 'outcome')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f647f62b5d0>
```



5. Build the model (use scikit learn)

```
[73] # Select the features based on correlation
Selected_features = ['rectal_temp', 'pulse', 'respiratory_rate', 'packed_cell_volume', 'total_protein', 'surgery_no', 'surgery_y

# Divide the dataset into dependent and independent variables
X = data2[Selected_features]
y = data2['outcome']
# Split the datasets into train and test data with testing data as 20%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)
```

```
# Import the modules needed for model building
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
# Define logistic regression model
clf = LogisticRegression(random_state=0, solver='lbfgs', multi_class='ovr', max_iter=10000)
clf = clf.fit(X_train, y_train)
Y_pred = clf.predict(X_test)
# Prediction
print(f"Prediction: {Y_pred}")
# Calculate the score
log_regr_score1 = clf.score(X_test, y_test)
print('Logistic Regression Score: ', round(log_regr_score1, 3))
# Evaluate the model using parameters such as precision, recall, f1-score, support
print(classification_report(y_test,Y_pred))
```

6. Summarize the model
7. Prediction
8. Evaluate the model

```
# Import the modules needed for model building
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
# Define logistic regression model
clf = LogisticRegression(random_state=0, solver='lbfgs', multi_class='ovr', max_iter=10000)
clf = clf.fit(X_train, y_train)
Y_pred = clf.predict(X_test)
# Prediction
print(f"Prediction: {Y_pred}")
# Calculate the score
log_regr_score1 = clf.score(X_test, y_test)
print('Logistic Regression Score: ', round(log_regr_score1, 3))
# Evaluate the model using parameters such as precision, recall, f1-score, support
print(classification_report(y_test,Y_pred))
```

```
Prediction: [0. 0. 1. 1. 1. 0. 1. 1. 1. 1.]
Logistic Regression Score:  0.7
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00         0
         1.0       1.00      0.70      0.82        10

    accuracy                           0.70        10
   macro avg       0.50      0.35      0.41        10
weighted avg       1.00      0.70      0.82        10
```

9. Tuning the model

```
# Tuning the parameters for logistic regresion
lr = LogisticRegression(random_state=0)
param_grid={"C":np.logspace(-3,3,10)}
grid = GridSearchCV(lr, param_grid, cv=5, verbose=0)
grid_search=grid.fit(X_train, y_train)
print('The best value found for the hyperparameter C is ' + str(grid_search.best_params_['C']))
print('Accuracy obtained after tuning the parameters: ' + str(grid_search.best_score_))
# Make prediction using the best parameters found in the grid search cv
y_pred = grid_search.predict(X_test)
```

```
The best value found for the hyperparameter C is 0.004641588833612777
Accuracy obtained after tuning the parameters: 0.8464285714285715
```

Part-B: Logistic Regression
Follow the general steps to carry out logistic regression as mentioned in Part-A.
Calculate the performance metrics-Accuracy, Miss-classification rate, Receiver operating characteristics.

Code:

1. Load the dataset (Use pandas) and libraries

```
# Importing the necessary libraries
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

```
[2] # Loading the dataset
train=pd.read_csv("./CreditScore_train.csv")
test=pd.read_csv("./CreditScore_test.csv")
```

2. Data Preprocessing (Handling NA values)

```
# Printing the shape of data set
train["source"] = "train"
test["source"] = "test"
print("Train Data Shape aftr adding target col : ",train.shape)
print("Test Data Shape aftr adding target col : ",test.shape)
```

```
Train Data Shape aftr adding target col :  (80000, 306)
Test Data Shape aftr adding target col :  (20000, 306)
```

```
[20] # Create the master dataset by concantenating training and test dataset
df = pd.concat([train,test])
df['y'] = 1/df['y']
df.head()
```

	x001	x002	x003	x004	x005	x006	x007	x008	x009	x010	x011	x012	x013	x014	x015	x016	x017	x018	x019	x020	x021	x022	x023	x024	x025	x026	x027	x028	x029	x030	x031	x032
0	1084094	426.0	39.0	128.0	426.0	0	0	0	0	0	2	4	4	9	19	5	14	8	5	6	2	3	2	1	1	1	1	6	1	5	0	0
1	1287777	160.0	2.0	64.0	160.0	1	1	2	0	1	3	3	7	5	21	5	16	5	3	9	2	3	1	9	1	1	1	3	9	6	0	0
2	1483016	163.0	16.0	104.0	239.0	0	0	0	1	0	1	0	0	6	8	3	5	4	2	7	3	3	2	0	0	0	1	0	0	4	1	0
3	959054	NaN	NaN	NaN	102.0	0	0	0	0	0	0	2	2	0	4	0	4	2	0	0	0	0	0	3	0	0	0	0	0	0	7	0
4	1342113	3.0	2.0	2.0	62.0	0	2	2	0	0	1	2	0	0	5	2	3	2	2	2	1	1	1	3	0	1	1	0	3	2	3	0

5 rows × 306 columns

```
[21] lst=[]
     cols=[]
     lst=df.columns
     row=df.shape[0]
     # Handle NA values by dropping the columns
     data=df.drop(cols,axis=1)
     data.shape
     print("Printing the number of null values in columns: ")
     print([cols.append(i) for i in lst if data[i].isnull().sum()/row*100 > 70])
     # Ensuring the data preprocessing

     Printing the number of null values in columns:
     [None, None, None, None, None, None, None, None, None, None]
```

```
[22] # Filtering the attributes based on its correlation with target value
     pd.options.display.max_rows = 4000
     colg=data.corr()['y'].sort_values() > 0.3
     coll=data.corr()['y'].sort_values() <-0.3
     data.shape

     (100000, 306)
```

```
[23] lstg=[]
     lstl=[]
     lstg.clear()
     lstl.clear()
     [lstg.append(i) for i,j in colg.items() if j == True]
     [lstl.append(i) for i,j in coll.items() if j == True]
     print("Length of lstl",len(lstl))
     print("Length of lstg",len(lstg))
     lstd=lstg+lstl
     print("Length of lstd",len(lstd))

     Length of lstl 37
     Length of lstg 29
     Length of lstd 66
```

```
[24] data_cols = data.columns
     cor_target = abs(data.corr()["y"])
     #Selecting highly correlated features
     relevant_features = cor_target[cor_target<0.3]
```

```
[25] lst_key = []
     for i,j in relevant_features.items():
         lst_key.append(i)
     lstd1= ['x017','x047','x015','x043','x251','x248','x018','x019','x028','x020','x004','x027','x030','x224','x260','x261','x229','x262','x247','x250','x246','x245','x014','x023','x002','x239','x0
     drop_cols=['x062','x066','x067','x068','x069','x070','x071','x072','x073','x074','x075','x076','x077','x078','x079','x080','x081','x082','x083','x084','x085','x086','x087','x088','x089','x090',
     data.shape

     (100000, 306)
```

```
[28] coL=list(prep_data.columns)
     type(coL)
```

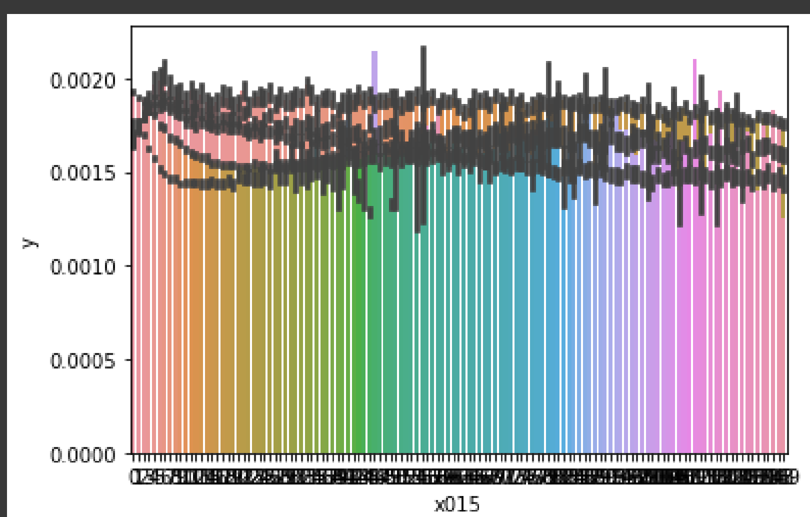
```
list
```

```
[29] # Dropping unnecessary columns from dataset
     train_test_b4_split_data=prep_data.copy
     train_final = prep_data[prep_data.source=="train"]
     test_final = prep_data[prep_data.source=="test"]
     train_final = train_final.drop(columns='source',axis=1)
     test_final = test_final.drop(columns='source',axis=1)
     train_final.columns

     # Cleaning the dataset (Checking for Nan values)
     def clean_dataset(df):
       assert isinstance(df,pd.DataFrame),"df needs to be a pd.DataFrame"
       df.dropna(inplace=True)
       indices_to_keep = ~df.isin([np.nan,np.inf,-np.inf]).any(1)
       return df[indices_to_keep].astype(np.float64)
     train_final = clean_dataset(train_final)
```

3. Exploratory Data Analysis (understanding the relationships between the variables with help of plot, scatter-plot, energy-plot etc) Use matplotlib

```
# Exploratory data analysis (Select features and plot them on a series)
cols = ['x002', 'x004', 'x005', 'x014', 'x015']
for a in cols:
  sns.barplot(data=prep_data, x=a, y='y')
```



4. Data Partition (80% for training and 20% for testing) (Use scikit-learn)

```
[30] X = train_final.drop("y",axis=1)
     Y = train_final["y"]
     print(X.shape)
     print(Y.shape)

     (18, 71)
     (18,)

[31] # Splitting the dataset as training and test dataset
     seed = 42
     test_size = 0.20
     X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=test_size)
     print(X_train.shape)
     print(X_test.shape)
     print(Y_train.shape)
     print(Y_test.shape)

     (14, 71)
     (4, 71)
     (14,)
     (4,)
```

5. Build the model (use scikit learn)
6. Summarize the model
7. Prediction
8. Evaluate the model

```
[ ] regr = LinearRegression()
    regr.fit(X_train,Y_train)
    print(regr.score(X_test,Y_test))

    0.784548662909031
```

Conclusion: As an introductory experiment in this lab, I handled various aspects of data such as handling NA values, normalization. I analyzed the data using python library to understand the data attributes and utilizing them to make predictions. I understood the logic behind linear and logistic regressions and implemented them on given problem statements. I observed significant accuracies and also tuned the parameters.