SIT384

Cyber security analytics

Portfolio Learning Summary Report

Kenisha Corera
DFCS|DK|62|203

Self-Assessment Details

The following checklists provide an overview of my self-assessment for this portfolio.

	Pass (D)	Credit (C)	Distinction (B)	High Distinction (A)
Self-Assessment	✓	✓	✓	✓

(Tick ✓ the box with the grade you are applying for)

(Check that you have included the minimum required details. Tick these boxes and ones for higher grades as applies.)

Self-Assessment Statement

	Included
Learning Summary Report	✓
Pass tasks complete	✓

Minimum Pass Checklist

	Included
All Credit Tasks are Complete on OnTrack	✓

Minimum Credit Checklist (in addition to Pass Checklist)

	Included
Distinction tasks are Complete	✓

Minimum Distinction Checklist (in addition to Credit Checklist)

	Included
High Distinction tasks are Complete	✓

Minimum High Distinction Checklist (in addition to Distinction Checklist)

Declaration

I declare that this portfolio is my individual work. I have not copied from any other student's work or from any other source except where due acknowledgment is made explicitly in the text, nor has any part of this submission been written for me by another person.

Signature: **J.K.Corera**

Portfolio Overview

Work that shows I've met every unit learning outcome for the SIT384 unit title with a high distinction level is included in this portfolio.

The work in this portfolio demonstrates that all of the Unit Learning Outcomes (ULOs) for the SIT384 Cyber Security Analytics unit have been met with a High Distinction. I knew a little bit about cybersecurity data types and analysis going into this unit. I learnt how to use a variety of data analytical methodologies to several real-world cybersecurity concerns as I developed. Additionally, I developed a great deal of experience designing and evaluating data-driven cybersecurity solutions with Python.

I have been learning about the many data formats utilised in cybersecurity systems, such as JSON, CSV, and various log formats, since the start of this course. With this assurance, I can now handle threat detection and network traffic by understanding and utilising these formats in both stored and sent data.

I have examined cybersecurity datasets using regression models, K-means clustering, and other classification techniques like as Logistic Regression and K-Nearest Neighbours. As a result, I was able to prove both technical and non-technical staff alike that I can effectively communicate complex ideas in a report or presentation by finishing multiple assignments.

Utilising Python to leverage complex data analytics solutions was one of the unit's biggest achievements. To collect and analyse data sets and create a method for separating and categorising security occurrences, I wrote code. I was able to demonstrate my problem-solving abilities when I evaluated these solutions and optimised them in reference to relevant libraries like Pandas, Scikit-learn, and Matplotlib.

I took sure to include details and justifications for my work in my portfolio so that I could assess its quality critically in comparison to the predetermined standards. I was able to demonstrate that I understood the tasks assigned to me in their entirety and that my answers were adequate by examining the accuracy and efficacy of the majority of my models. I should add that maintaining this self-control was essential to preserving my peak performance at work.

Altogether, this portfolio displays a thorough comprehension of the subject of cyber security analytics. I think that my success in getting a High Distinction was a result of my ability to stay focused, observe how and what I was teaching by going beyond the course unit, and employ fresh ways.

Reflection

The most important things I learnt:

I believe that the two most significant things I learnt in this unit were the real-world applications of machine learning techniques for cybersecurity using K-Nearest Neighbours

and logistic regression on authentic datasets. Additionally, I have a solid understanding of how data type affects security and how Python can be used to filter and alter various forms.

The things that helped me most were:

It was discovered that a directory for reviewing Python programming tutorials and data analytics was quite helpful. Classification and clustering were the most beneficial exercises for me because they helped me remember the theory underlying each tactic and gave me some practical experience using various replies.

I found the following topics particularly challenging:

The present unit's two most difficult components were fine-tuning the parameters of machine learning models, particularly Random Forest and K-Nearest Neighbours. I occasionally had trouble determining what the best parameters would be, but as I kept practicing and learnt more about the subject, I was able to increase my model's efficacy.

I found the following topics particularly interesting:

Regarding the methods, the clustering algorithms—and the K-means technique in particular—piqued my curiosity. The fact that seeming equals may be categorised without the labels needing to be considered previously or predictively startled me. This has consequences for identifying cybersecurity threats.

I feel I learnt these topics, concepts, and/or tools really well:

I am certain that my understanding of Using Python for Data Analytics has not been compromised, since I have mastered the application of clustering and classification ideas through my use of Scikit-learn. In addition, I gained proficiency in the analysis and visualization of security data, which was essential for accurately completing the unit's assignments.

I still need to work on the following areas:

Gaining a deeper understanding of deep learning and its relationship to cybersecurity data is my goal here. Though I now have a solid foundation in "shallow learning," I am not pleased with the depth of the work, and I plan to focus more on deep learning since it may provide more complex solutions to the threat detection problem.

This unit will help me in the future:

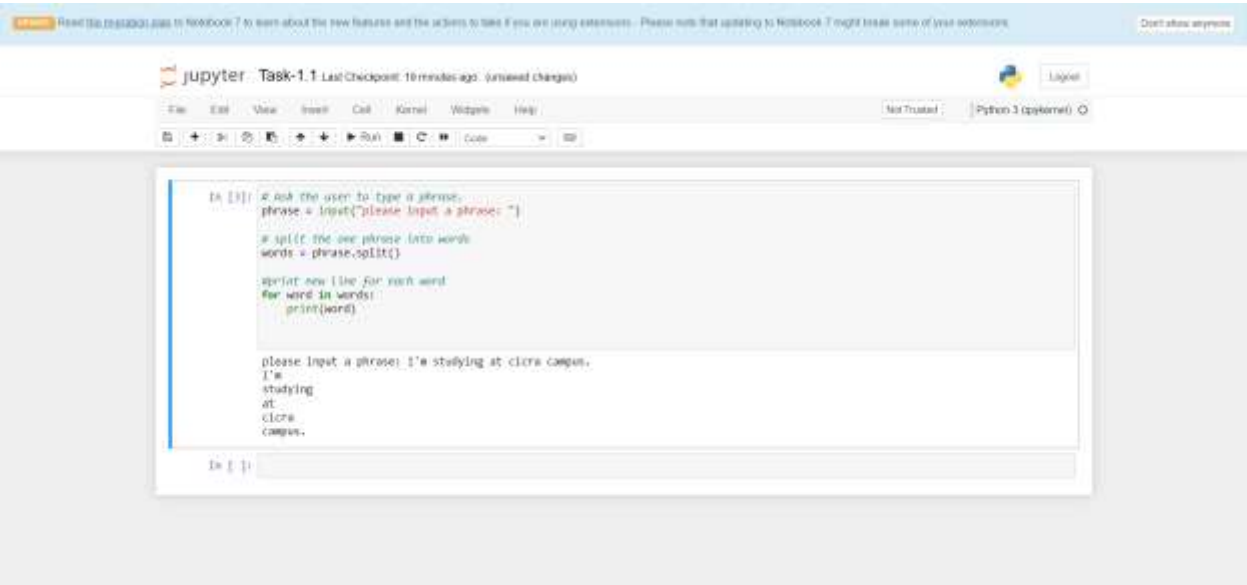
After this session, I will be able to use these abilities to my next job as a cybersecurity analyst. To better understand and handle cyber threats, security data needs to be thoroughly analyzed and then categorized. When making decisions based on data, advanced security analytics jobs can benefit from knowledge of Python and machine learning methodology.

If I did this unit again I would do the following things differently:

If I could redo this unit, I would start by studying the theory behind each algorithm rather than attempting to implement them one at a time. That would enable me to complete the tasks with a deeper comprehension and might even result in the early development of better models.

Task 1.1 P

Get a phrase from user, split the words of that phrase and print the words.



Task 1.2P

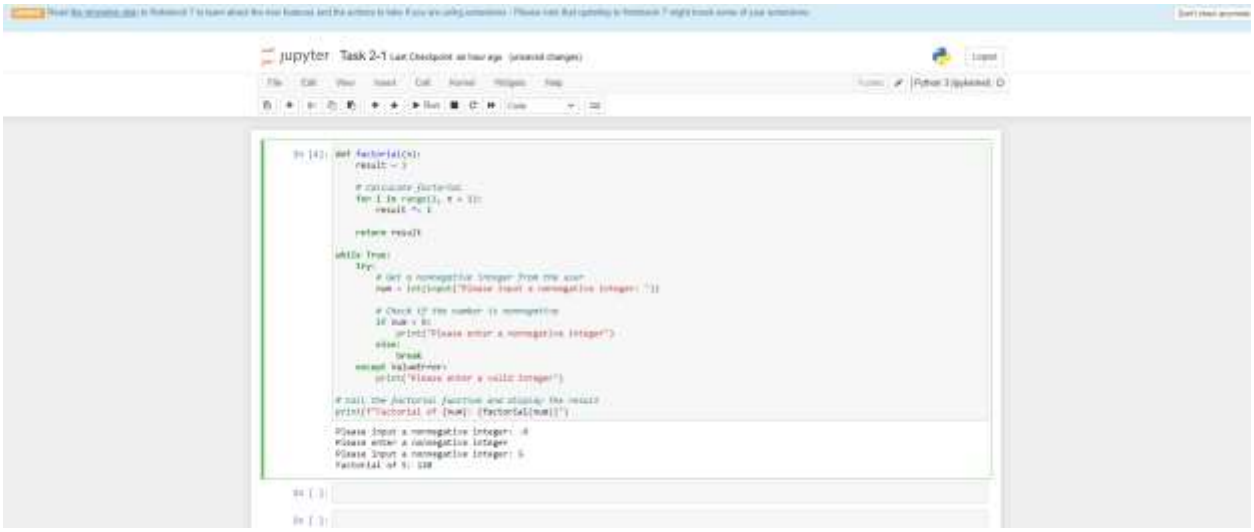
Get an integer from user, draw a square of "*" based on the value of the input. If the input is not a positive integer, prompt error and require a new input.

In [1]:
```
while True:
    try:
        # Get an integer from the user
        n = int(input("Please input a positive integer: "))

        # Check if the input is a positive integer
        if n <= 0:
            print ("Integer should be greater than 0")
        else:
            break
    except ValueError:
        print("Sorry, invalid input. Please enter an integer")

for i in range(n):
    print("*" * n)
```

```
Please input a positive integer: -1
Integer should be greater than 0
Please input a positive integer: in
Sorry, invalid input. Please enter an integer
Please input a positive integer: 13
*************
*************
*************
*************
*************
*************
*************
*************
*************
*************
*************
*************
*************
```

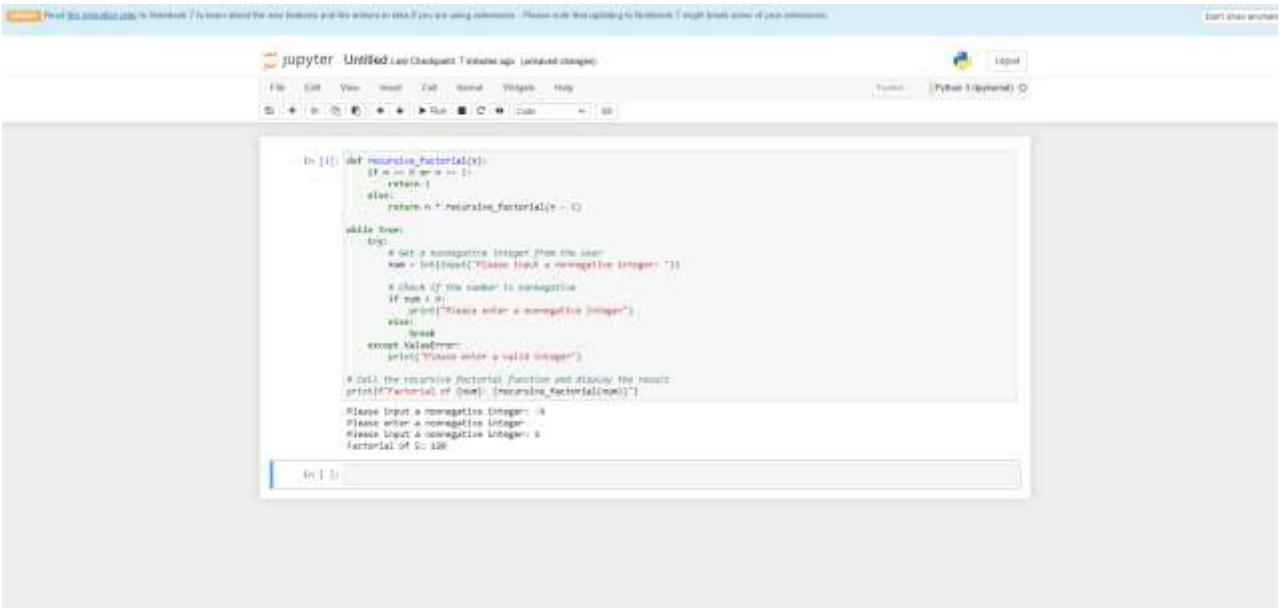
Task 2.1

Define a function which accepts a passed argument and calculates its factorial. A program accepts user's input and calls the function. (Please DO NOT use recursive function call in the function definition.) (Sample output as shown in the following figure is for demonstration purposes only.)



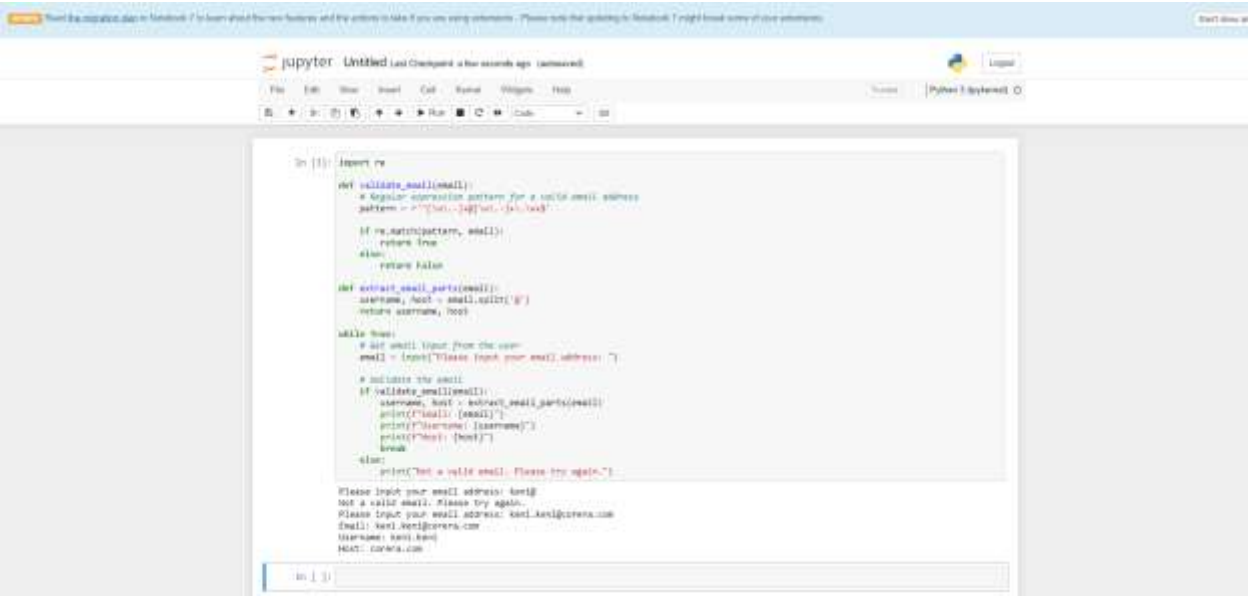
Task 2.2P

Define a function which accepts a passed argument and calculates its factorial. A program accepts user's input and calls the function. (Please use recursive function call in the function definition.) (Sample output as shown in the following figure is for demonstration purposes only.)



Task 2.3P

Use regular expression to implement a function which accepts a passed string and check if it is a valid email address. A program accepts user's input and calls the email validation function. If the email is valid, print out the email, username and host. (Rule: word characters (a-zA-Z0-9_ or "\w"), dot, and hyphen are considered as valid characters in an email address, besides of one & only one '@'.)

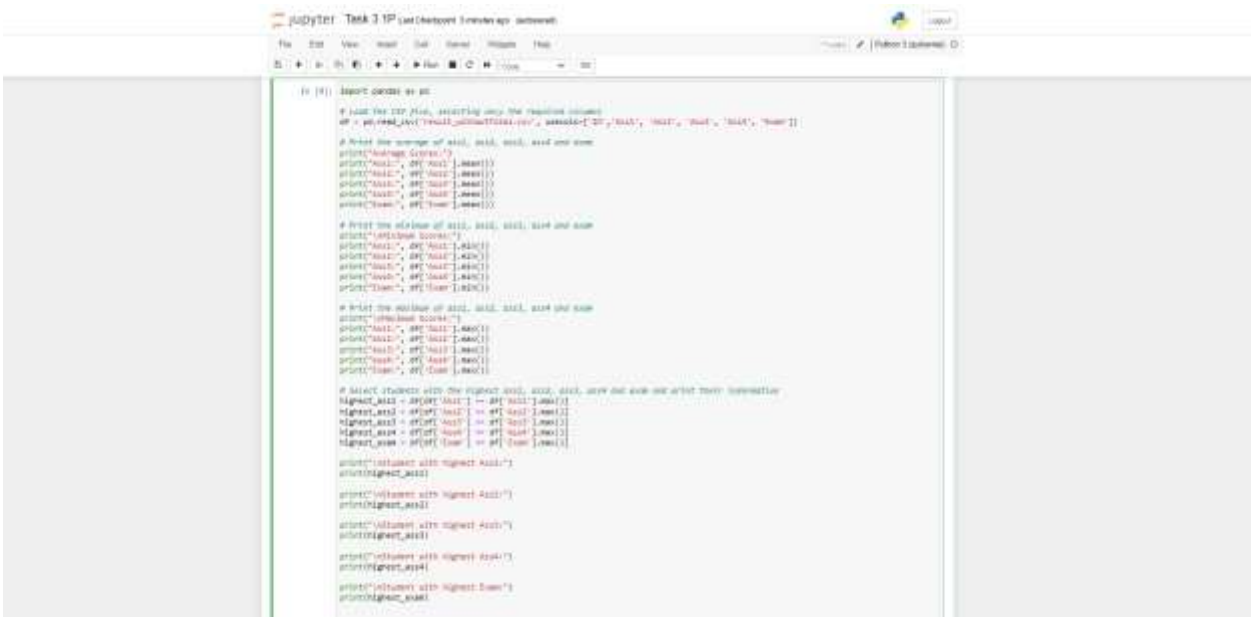


Task 3.1P Process data from file

- ✓ You are given a student result data file (result_withoutTotal.csv). It has columns: ID: student id Ass1 ~ Ass4: assignment scores (out of 100); weight of ass1, ass2, ass3 and ass4 is 5%, 15%, 5%, and 15%, respectively. Exam: examination score (out of 120); weight is 60%.

- Print average of ass1, ass2, ass3, ass4 and exam column, respectively.
- Print min of ass1, ass2, ass3, ass4 and exam column, respectively.
- Print max of ass1, ass2, ass3, ass4 and exam column, respectively.
- Select the students with the highest ass1, ass2, ass3, ass4 and exam, respectively, and print their information (ID, Ass1, Ass2, …, Exam)

- This Python script reads a CSV file of student results and performs various statistical analyses. It imports the Pandas library, reads the file, and selects columns ID, Ass1, Ass2, Ass3, Ass4, and Exam. It then calculates and prints the average, minimum, and maximum scores for each assignment and the exam. The script identifies the student(s) with the highest scores in each category by filtering the DataFrame based on the maximum values. It prints the complete information of these top-performing students. This analysis helps educators quickly understand overall trends and identify individual high achievers, aiding in performance assessment and decision-making.



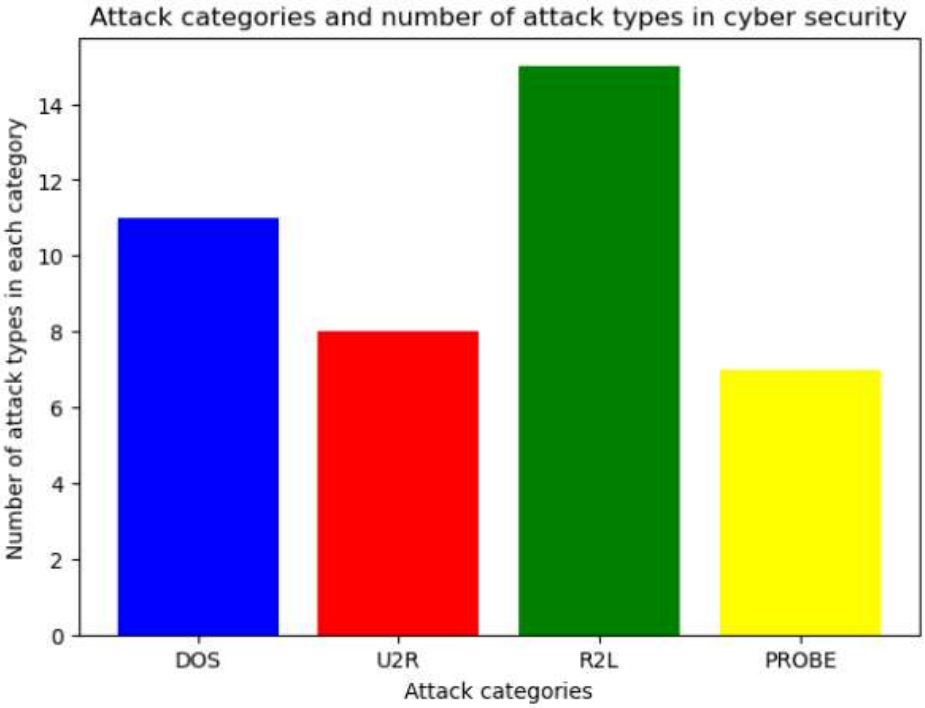
```
Average Scores:
Ass1: 79.08241758241758
Ass2: 72.01078021978022
Ass3: 71.76373626373626
Ass4: 83.7054945054945
Exam: 57.24175824175824

Minimum Scores:
Ass1: 50.0
Ass2: 43.8
Ass3: 0.0
Ass4: 0.0
Exam: 0

Maximum Scores:
Ass1: 99.1
Ass2: 100.0
Ass3: 98.0
Ass4: 100.0
Exam: 106

Student with highest Ass1:
    ID  Ass1  Ass2  Ass3  Ass4  Exam
52  53  99.1  86.6  82.0  93.1    57

Student with highest Ass2:
    ID  Ass1   Ass2  Ass3   Ass4  Exam
10  11  96.6  100.0  98.0   97.3   102
49  50  93.6  100.0  97.0  100.0    90
83  84  93.6  100.0  96.0  100.0   106

Student with highest Ass3:
    ID  Ass1   Ass2  Ass3  Ass4  Exam
10  11  96.6  100.0  98.0  97.3   102
40  41  88.0   58.5  98.0  91.8    71

Student with highest Ass4:
    ID  Ass1   Ass2  Ass3   Ass4  Exam
33  34  94.4   97.5  80.0  100.0    90
34  35  92.7   76.8  84.0  100.0    55
49  50  93.6  100.0  97.0  100.0    90
83  84  93.6  100.0  96.0  100.0   106
87  88  96.0   97.6  65.0  100.0    76

Student with highest Exam:
    ID  Ass1   Ass2  Ass3   Ass4  Exam
83  84  93.6  100.0  96.0  100.0   106
```

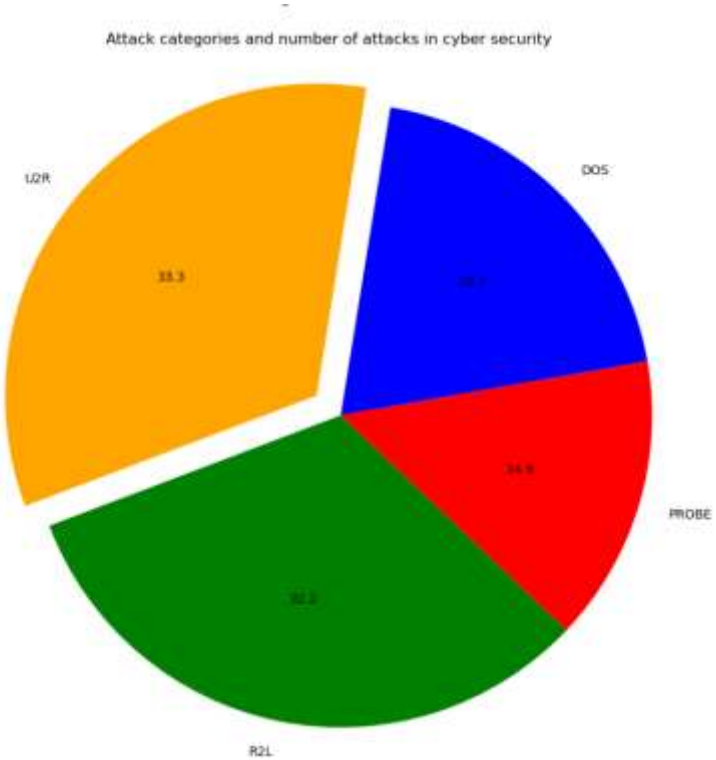
Task 4.1P

Visualize data using matplotlib

Pie chart and bar chart creation with `matplotlib` and data processing with `pandas` were used in the cybersecurity attack data visualization work. Functions like groupby(), sum(), and value_counts() must be understood in order to load the data and group it to count attack kinds. A clear comparison of the number of attack types across categories was shown by the bar chart, while the distribution of assault categories was effectively displayed by the pie chart, with the largest portion emphasized. By inferring labels directly from the data, the initial labeling and value issues were rectified. The practical uses of these abilities in cybersecurity and other data-driven professions were highlighted, along with the significance of proper data representation, in this exercise. Developing my knowledge of data processing techniques and investigating sophisticated visualization tools are among my future objectives.

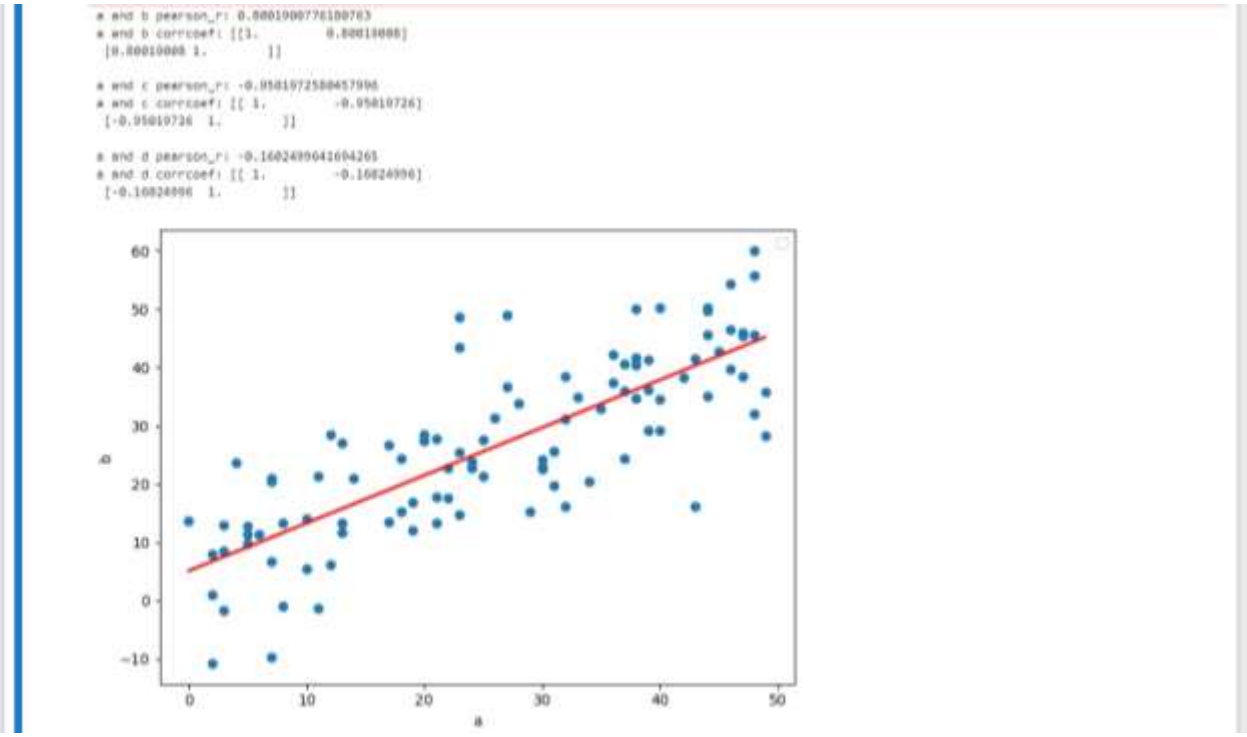
In [28]:
```
# Here I have read the  .csv file name
file_name = 'attack-type-frequency.csv'
data = pd.read_csv(file_name, index_col=0, engine='python')

#Here I have used head to display the first five rows of the data frame.
data.head()

# Group by attack category and count the number of attack types and sum of attacks
attack_types_count = data.groupby('category').size()
attack_numbers_sum = data.groupby('category')['number_of_attack'].sum()

# Define the colors and labels
colors_bar = ['blue', 'red', 'green', 'yellow']  # Colors for the bar chart
colors_pie = ['blue', 'orange', 'green', 'red']  # Colors for the pie chart
labels = ['DOS', 'U2R', 'R2L', 'PROBE']

# Ensure the order of attack_types_count matches the labels order
attack_types_count = attack_types_count.reindex([label.lower() for label in labels])

# Bar chart for number of attack types in each category
plt.figure(figsize=(7, 5), dpi=100)
plt.bar(labels, attack_types_count, color=colors_bar)
plt.xlabel('Attack categories')
plt.ylabel('Number of attack types in each category')
plt.title('Attack categories and number of attack types in cyber security')
plt.show()

# Ensure the order of attack_numbers_sum matches the labels order
attack_numbers_sum = attack_numbers_sum.reindex([label.lower() for label in labels])

# Define the custom autopct function
def absolute_value(val):
    return f'{val:.1f}'

# Pie chart for number of attacks in each category
explode = [0.1 if x == max(attack_numbers_sum) else 0 for x in attack_numbers_sum]

plt.figure(figsize=(10, 10))
plt.pie(attack_numbers_sum, labels=labels, colors=colors_pie, autopct=absolute_value, explode=explode, startangle=10)
plt.title('Attack categories and number of attacks in cyber security')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```

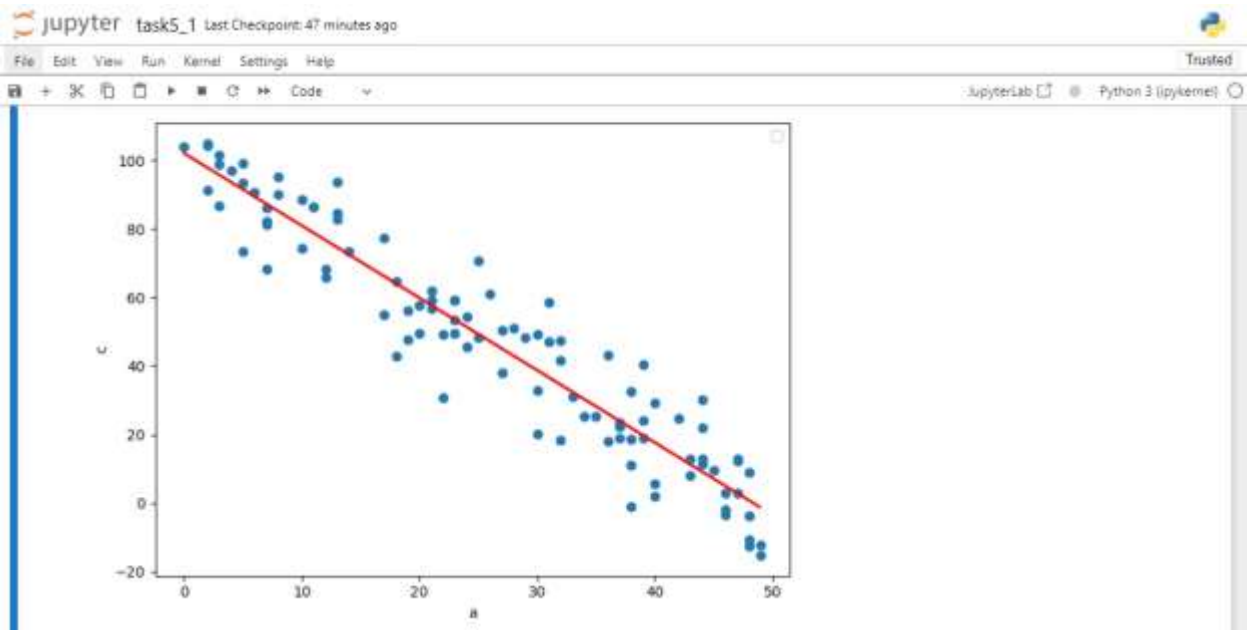

Attack categories and number of attack types in cyber security
Number of attack types in each category
0
2
4
6
8
10
12
14
DOS
U2R
R2L
PROBE
Attack categories

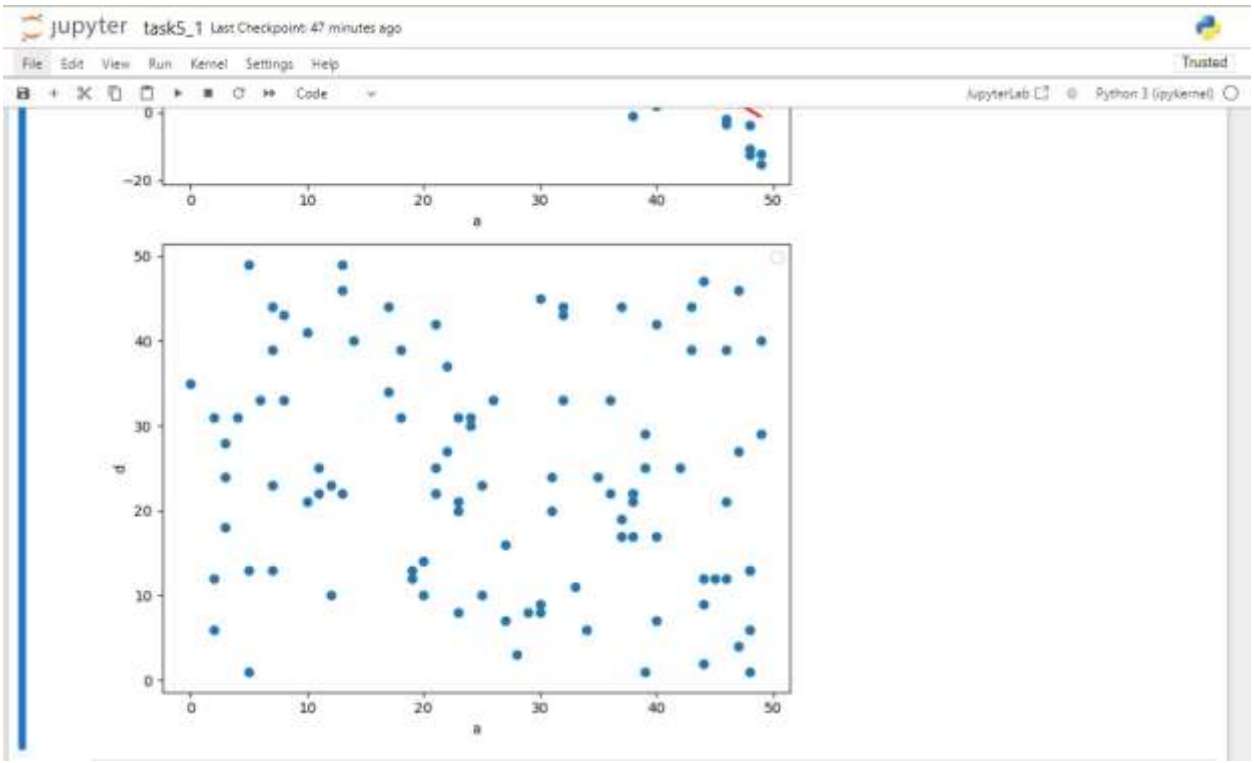

Attack categories and number of attacks in cyber security
U2R
DOS
PROBE
R2L

Pass Task 5.1P: Data correlation

Numpy implements a corrcoef() function that returns a matrix of correlations of x with x, x with y, y with x and y with y. We're interested in the values of correlation of x with y (so position (1, 0) or (0, 1)). The values of correlation of x with y might be Positive Correlation, Negative Correlation, or No/Weak Correlation.







The program examines the relationships among three variables ('b, 'c, and 'd') and the target variable ('a') in a CSV dataset. The correlation coefficient matrices and the results of the Pearson's correlation coefficient calculations are printed together.

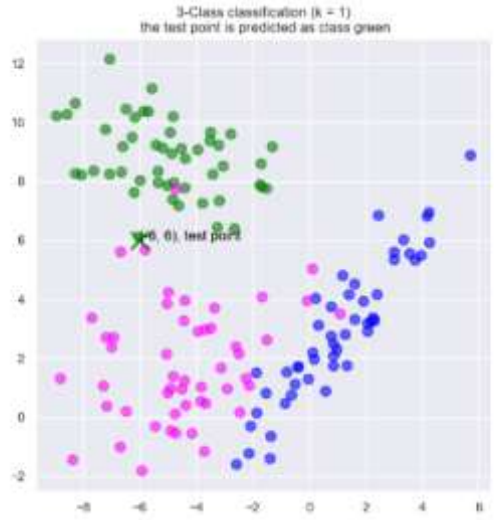
The correlation matrix and Pearson correlation coefficient for each pair of variables ('a' and 'b', 'a' and 'c', 'a' and 'd') are calculated and printed using the print_correlation function.

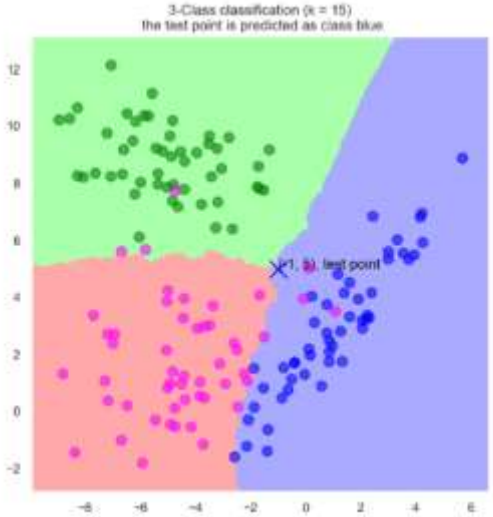
Every pair of variables with 'a' on the x-axis receives a scatter plot from the plot_correlation function, which also, if the absolute correlation coefficient is higher than 0.5, may fit a line of best fit.

For every pair of variables, a subplot is made, and a line of best fit is optionally included in the scatter plots. Plt.tight_layout is used to provide appropriate spacing between the subplots, and plt.show is used to render the plots.

6.1P: kNN classification

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#%matplotlib inline
from matplotlib.colors import ListedColormap
from sklearn import neighbors

sns.set()


dataset=pd.read_csv('task6_1_dataset.csv')
x1=dataset['x1']
x2=dataset['x2']
X_train = np.c_[x1, x2]

Y_train = dataset['y'].values
print(type(Y_train))

k = 1
knn = neighbors.KNeighborsClassifier(k)
knn.fit(X_train, Y_train)

cmap_bold = ListedColormap(['green', 'blue', 'magenta'])

fig,ax = plt.subplots(figsize=(7, 7), dpi=100)

scale = 75
alpha = 0.6
colors = ['green', 'blue', 'magenta']

ax.scatter(X_train[:, 0], X_train[:, 1], c=Y_train, cmap=cmap_bold, alpha=alpha, s=scale)

X_test = [[-6,6]]
Y_pred = knn.predict(X_test)

ax.scatter(X_test[0][0], X_test[0][1], marker="x", s=3*scale, lw=2, c=colors[Y_pred.astype(int)[0]])


ax.set_title("3-Class classification (k = {})\n the test point is predicted as class {}".format(k, colors[Y_pred.astype(int)[0]]
ax.text(X_test[0][0], X_test[0][1], '(%d, %d), %s' % (X_test[0][0], X_test[0][1],'test point') , color='black')


k = 15
knn = neighbors.KNeighborsClassifier(k)
knn.fit(X_train, Y_train)

h = 0.05

cmap_light = ListedColormap(['#AAFFAA', '#AAAAFF', '#FFAAAA'])

x1_min, x1_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
x2_min, x2_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, h), np.arange(x2_min, x2_max, h))

Z = knn.predict(np.c_[xx1.ravel(), xx2.ravel()])


Z = Z.reshape(xx1.shape)

fig,ax = plt.subplots(figsize=(7, 7), dpi=100)
ax.pcolormesh(xx1, xx2, Z, cmap=cmap_light)

ax.scatter(X_train[:, 0], X_train[:, 1], c=Y_train, cmap=cmap_bold, alpha=alpha, s=scale)

plt.xlim(xx1.min(), xx1.max())
plt.ylim(xx2.min(), xx2.max())


X_test = [[-1, 5]]
Y_pred = knn.predict(X_test)
ax.scatter(X_test[0][0], X_test[0][1], alpha=0.95, color=colors[Y_pred.astype(int)[0]], marker='x', s=3*scale)

ax.set_title("3-Class classification (k = {})\n the test point is predicted as class {}".format(k, colors[Y_pred.astype(int)[0]]

ax.text(X_test[0][0], X_test[0][1], '(%d, %d), %s' % (X_test[0][0], X_test[0][1],'test point'), color='black' )
```





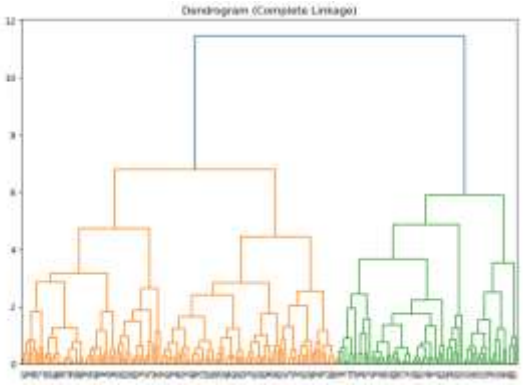
This Python software uses a fictitious dataset with two features (x1 and x2) to illustrate k-Nearest Neighbors (k-NN) classification. The script starts by importing the required libraries, which include pandas for data processing, matplotlib and seaborn for visualization, numpy for numerical computations, and sklearn for the k-NN method.

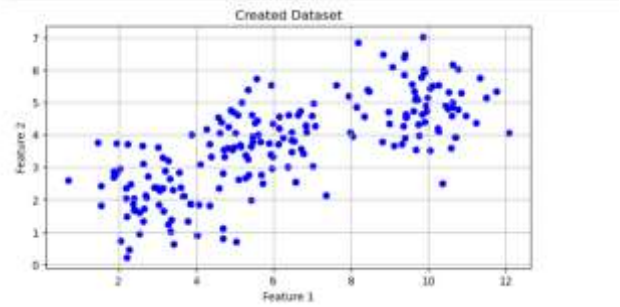
The dataset is loaded into a pandas Data Frame, where the class labels are stored in Y_train and the features are stored in X_train. When the k-NN classifier is originally configured with k=1, it predicts a test point's class by comparing it to its nearest neighbor in the training set. A scatter plot is used to show the training data, with distinct colors used for each class. The predicted class of the test point is highlighted.

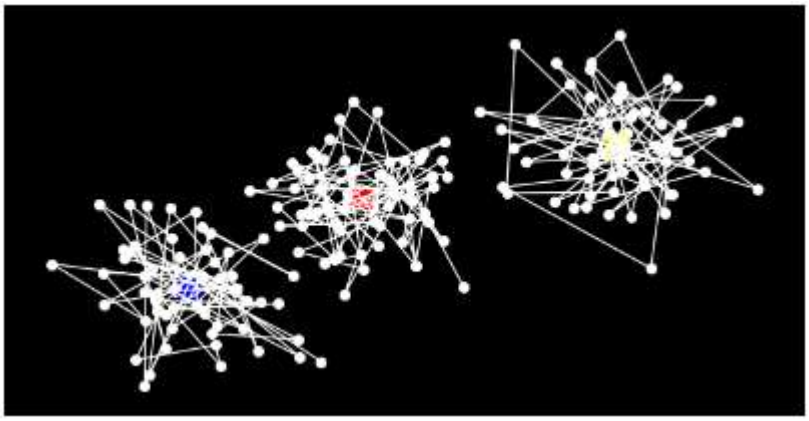
After that, the script raises k to 15 and retrains the model, producing a fine grid that shows how the decision boundaries alter as k increases. To demonstrate the impact of having more neighbors on the classification outcome, another test point is classified and shown. This method demonstrates how k-NN functions and how changing k affects the model's bounds for decisions and predictions.

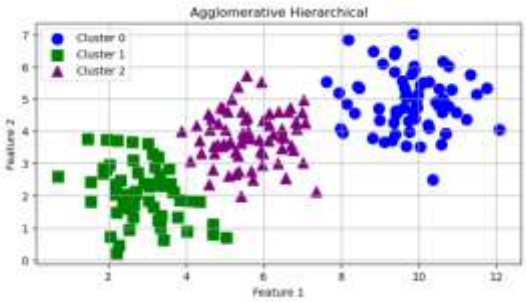
Understanding how the algorithm classifies fresh data points according to their proximity to already labeled data is made simpler by the display. It also emphasizes how crucial it is to select the right k value in order to strike a compromise between decision boundary smoothness and model sensitivity.

Pass Task 7.1P: K-Means and Hierarchical Clustering









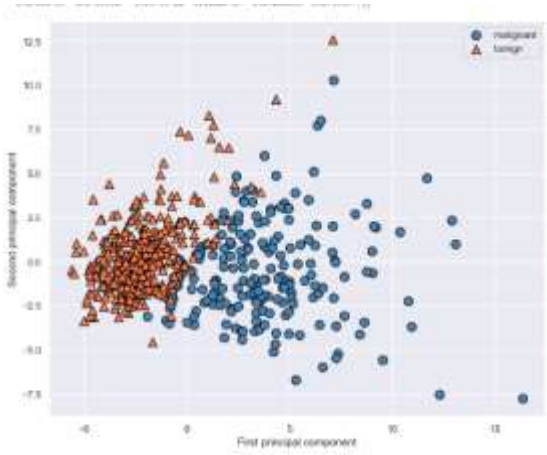
Two crucial unsupervised learning algorithms—K-Means and Agglomerative Hierarchical clustering—are applied practically in this Python code. The method first uses `make blobs` to create a synthetic dataset, from which it generates 200 data points distributed among three different clusters. The dataset is subsequently split into three clusters using the K-Means algorithm, with the centroids prominently shown. This illustrates how K-Means iteratively improves cluster centers. The distinct visual representation, including colored clusters on a black background, highlights the algorithm's capacity to efficiently group related data points.

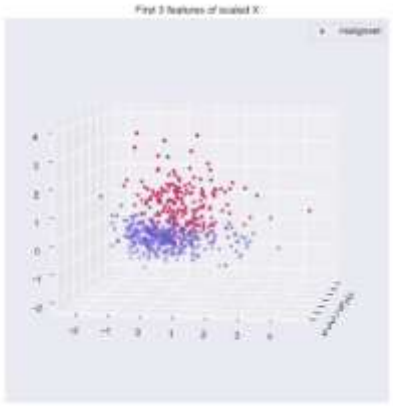
Afterwards, Agglomerative Hierarchical clustering is implemented by the code, which merges the nearest clusters one after the other to create a hierarchical tree. The whole linkage approach used to compute the distance matrix is projected on a dendrogram, providing a detailed perspective of the clustering hierarchy. The resulting clusters are visualized with unique markers. The relevance of data visualization in evaluating machine learning results is emphasized by this activity, which gives a practical grasp of both clustering algorithms. The understanding of these potent analytical tools is further improved by reinforcing the ideas of cluster centroids, hierarchical linkages, and the significance of linkage criteria in hierarchical clustering.

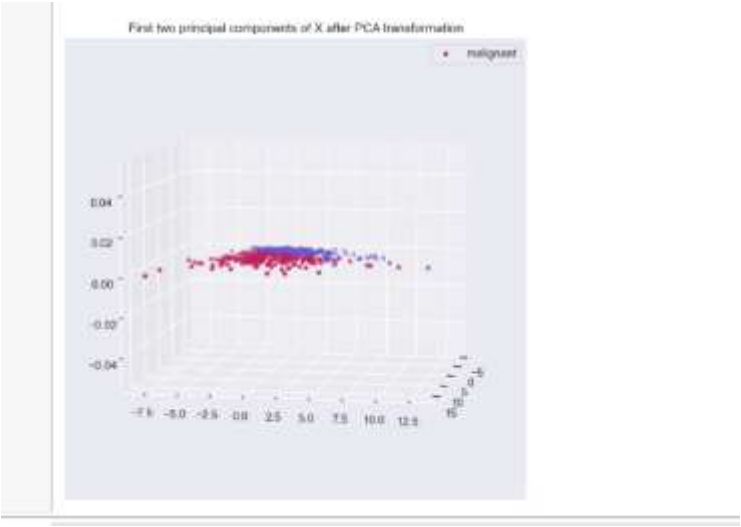
8.1P - PCA dimensionality reduction

PCA (Principle Component Analysis) is a dimensionality reduction technique that projects the data into a lower dimensional space. It can be used to reduce high dimensional data into 2 or 3 dimensions so that we can visualize and hopefully understand the data better.

In this task, you use PCA to reduce the dimensionality of a given dataset and visualize the data.







This code uses the Scikit-learn library's Breast Cancer dataset to conduct Principal Component Analysis (PCA) and displays the findings in both 2D and 3D graphs.

First, the load_breast_cancer function is used to load the dataset. This method returns information about cases of breast cancer, including characteristics that can be used to identify a tumour as benign or malignant. Next, to guarantee that every feature contributes equally to the analysis—a crucial component of PCA—the data is standardised using Scikit-learn's StandardScaler.

In order to visualise the data in two dimensions while retaining as much variation as feasible, PCA is used to reduce the dataset's dimensionality from its initial 30 features down to 2 primary components. To determine the success of the reduction, the code prints the principle components, the PCA component matrix, and the shapes of the original and reduced datasets.

To visualise these two main components, a 2D scatter plot is made, with various colours denoting benign and malignant cases. Benign instances are displayed with coral-colored triangular markers, whereas malignant cases are represented with blue circular markers. Based on the first two main components, which account for the majority of the variance in the data, this figure facilitates the observation of the differences between the two groups of tumours.

Subsequently, the algorithm generates two 3D scatter plots and specifies a custom colour map for visualisation. A more thorough understanding of the distribution of the data in three dimensions is made possible by the first 3D graphic, which displays the first three aspects of the standardised dataset. The data is once more visualised using the two principle components that were obtained using PCA in the second 3D graphic. By changing the viewing angle and viewpoint, these 3D visualisations aim to make the structure and separation of data points easier to interpret.

Overall, this code shows how PCA can effectively analyse and interpret a complicated dataset by reducing its dimensionality while preserving important information. It does this by combining 2D and 3D visualisations.

Pass Task 9.1P: Grid Search with Cross-Validation

In [2]:
```
# Import necessary libraries
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

# Load the dataset
digits = load_digits()
X = digits.data
y = digits.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Define the parameter grid
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}

# Initialize the model
svc = SVC()

# Perform grid search with cross-validation
grid_search = GridSearchCV(svc, param_grid, cv=5, return_train_score=True)
grid_search.fit(X_train, y_train)

# Print results
print(f"Test Score: {grid_search.score(X_test, y_test)}")
print(f"Best Parameters: {grid_search.best_params_}")
print(f"Best Cross-Validation Score: {grid_search.best_score_}")
print(f"Best Estimator: {grid_search.best_estimator_}")
```

```
Test Score: 0.9916666666666667
Best Parameters: {'C': 1, 'gamma': 0.001}
Best Cross-Validation Score: 0.9909528648857917
Best Estimator: SVC(C=1, gamma=0.001)
```

In [ ]:

Using grid search and cross-validation on the popular digits dataset from the scikit-learn module, this Python method tunes the hyperparameters of a Support Vector Machine (SVM) classifier. The code starts by importing the dataset and dividing it into training and testing sets. A random state is then set to ensure reproducibility. Using a parameter grid, the best values for two critical hyperparameters—gamma, which defines the influence of a single training example, and C, which regulates the trade-off between a smooth decision boundary and accurately classifying training points—are found in order to maximize the performance of the SVM model.
The various combinations of C and gamma are then assessed using a grid search with 5-fold cross-validation, with the combination that produces the highest cross-validation score being chosen in the end. The code provides a thorough picture of the model's tuning procedure and ultimate performance by printing out the test score, optimal parameters, best cross-validation score, and best estimator. This method guarantees that the SVM model retains its high accuracy while generalizing well to new data.

Task 3.2C Process data using Pandas

- ✓ You are given a student result data file (result_withoutTotal.csv). It has columns: ID: student id Ass1 ~ Ass4: assignment scores (out of 100); weight of ass1, ass2, ass3 and ass4 is 5%, 15%, 5%, and 15%, respectively. Exam: examination score (out of 120); weight is 60%. Total score can be calculated using formula: Total = 5%*(ass1+ass3) + 15%*(ass2+ass4) + 50%*exam (as exam is out of 120) Read students' result data from file result_withoutTotal.csv,

- Total column: Total = 5%*(ass1+ass3) + 15%*(ass2+ass4) + 50%*exam.
- Final column: Final = Total score rounded to the nearest integer.
- To pass the unit, a student must achieve at least 50 of the Total and 40% of Exam which is 48 out of 120 (or Total >= 50 and Exam >= 48).
- If a student failed the hurdle (Exam >= 48), the max Final is 44. No change to Final score if Final
- Grade column: N (Final <=49.45), P (49.45 < Final <=59.45), C (59.45 < Final <=69.45), D (69.45 < Final <=79.45) and HD (79.45 < Final). Border values are as follows:

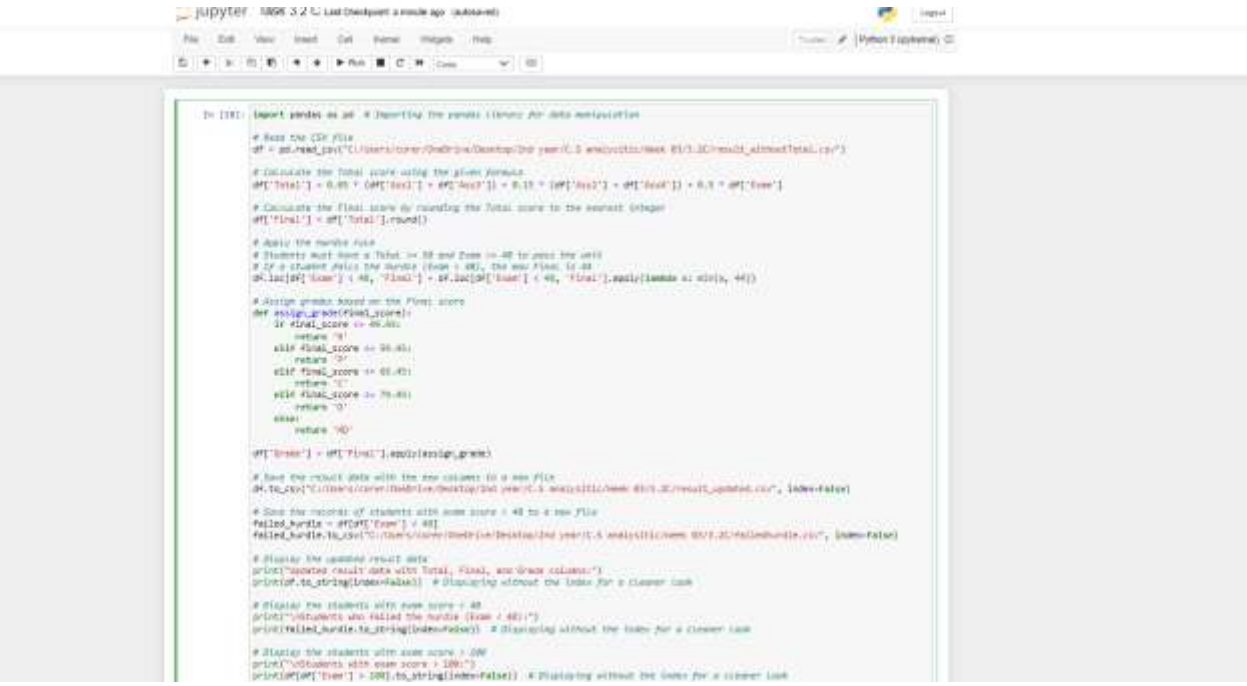
Save

- the result data file with the 3 new columns to a file called result_updated.csv.
- the students' records with exam score < 48 to a file called failedhurdle.csv

Display

- the result data file with the 3 new columns
- the students with exam score < 48 (these who failed the hurdle)
- the students with exam score > 100

- This Python script processes student results from a CSV file to calculate total and final scores, apply a hurdle rule, and assign grades. It uses Pandas to read the CSV file and compute the total score based on assignment and exam weights, then rounds it to create the final score. The script enforces a hurdle rule where students must score at least 50 overall and 48 in the exam to pass, capping the final score at 44 for those who fail the exam. It assigns grades based on the final score and saves the updated data to a new CSV file. Additionally, it saves and displays records of students failing the hurdle and those with exam scores above 100.



```
Updated result data with Total, Final, and Grade columns:
 ID  Ass1  Ass2  Ass3  Ass4  Exam  Total  Final Grade   total
  1  89.1  50.0  85.0  88.9    65 62.040   62.0     C  68.540
  2  95.1  82.5  90.5  94.5    52 61.830   62.0     C  67.030
  3  74.3  54.4  63.0  63.9    31 40.110   40.0     N  43.210
  4  89.8  81.3  82.0  90.4    37 52.845   44.0     N  56.545
  5  91.3  98.8  92.5  95.9    79 77.895   78.0     D  85.795
  6  83.9  82.5  89.0  98.6    68 69.810   70.0     D  76.610
  7  81.9  50.0  68.5  95.4    59 58.830   59.0     P  64.730
  8  50.0  54.9  50.0  87.7    51 51.890   52.0     P  56.990
  9  90.5  65.9  50.0  72.2    63 59.240   59.0     P  65.540
 10  89.0  89.9  94.0  90.3    84 78.180   78.0     D  86.580
 11  96.6 100.0  98.0  97.3   102 90.325   90.0    HD 100.525
 12  58.8  67.5  72.0  75.4    51 53.475   53.0     P  58.575
 13  74.7  78.5  50.0  86.2    66 63.940   64.0     C  70.540
 14  87.7  80.0  57.0  73.6    52 56.275   56.0     P  61.475
 15  66.3  53.7  53.0  81.9    30 41.305   41.0     N  44.305
 16  82.7  77.2  73.0  95.9    50 58.750   59.0     P  63.750
 17  97.5  96.2  83.5  93.8    55 65.050   65.0     C  70.550
```

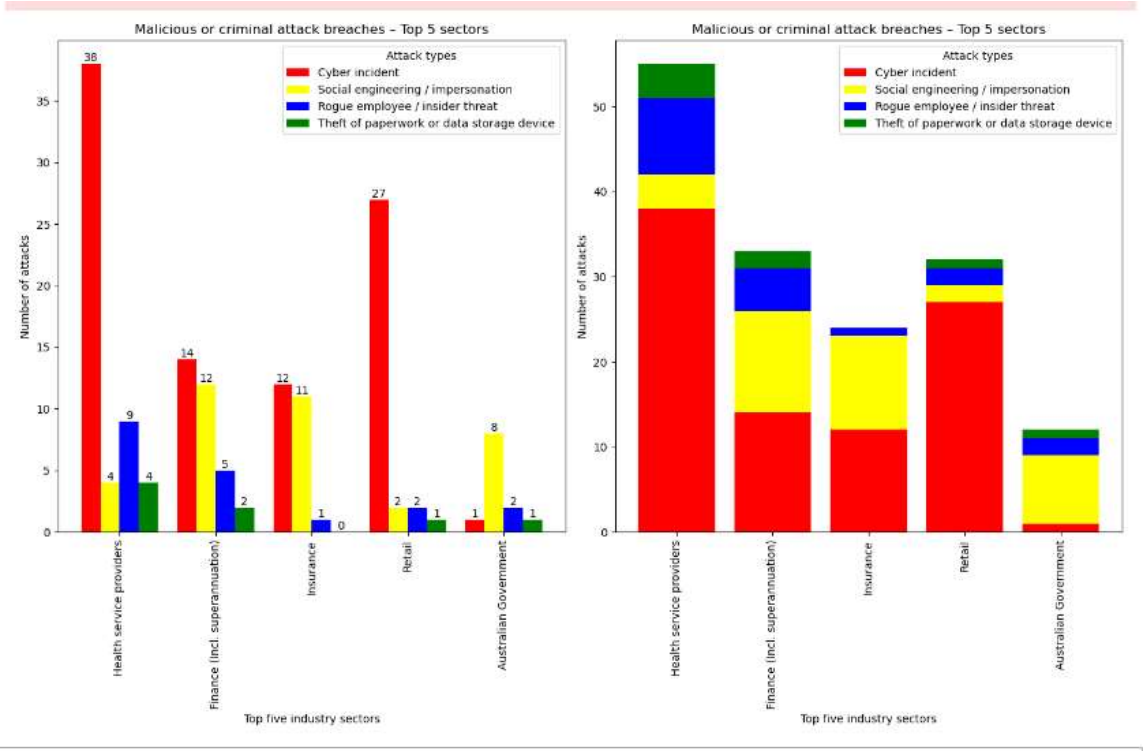
```
 ID  Ass1  Ass2  Ass3  Ass4  Exam  Total  Final Grade  total
  3  74.3  54.4  63.0  63.9    31 40.110   40.0     N 43.210
  4  89.8  81.3  82.0  90.4    37 52.845   44.0     N 56.545
 15  66.3  53.7  53.0  81.9    30 41.305   41.0     N 44.305
 24  57.7  76.3  71.0  87.7    35 48.535   44.0     N 52.035
 25  84.7  65.0  73.0  88.9    34 47.970   44.0     N 51.370
 26  84.7  53.8  75.0  78.1    36 45.770   44.0     N 49.370
 33  64.2  50.0  18.0   0.0     0 11.610   12.0     N 11.610
 42  81.5  43.8   0.0   0.0     0 10.645   11.0     N 10.645
 44  71.9  61.3  76.0  94.5    38 49.765   44.0     N 53.565
 47  50.0  71.3  56.0  93.8    34 47.065   44.0     N 50.465
 54  76.1  50.0  50.0  50.0    33 37.805   38.0     N 41.105
 60  73.9  53.2  74.0  95.9    34 46.760   44.0     N 50.160
 78  52.9  53.2  50.0  50.0    36 38.625   39.0     N 42.225
```

Task 4.2C

Visualize data with grouped bar chart and stacked bar chart

Two bar charts are produced by the code to show malevolent or illegal attacks on five different industry sectors. For clarity, different colors are used to compare assault kinds inside each sector in the grouped bar chart. The stacked bar chart stacks various attack kinds to highlight the cumulative impact of each attack type and displays the total number of attacks per sector. Both charts feature efficient labeling and color coding to provide a clear and thorough perspective of the data.

In [5]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Read the CSV file into a DataFrame
file_name = 'Malicious_or_criminal_attacks_breakdown-Top_five_sectors_July-Dec-2023.csv'
df = pd.read_csv('Malicious_or_criminal_attacks_breakdown-Top_five_sectors_July-Dec-2023.csv', index_col=0, engine='python')

# Data for plotting
sectors = df.columns
attack_types = df.index
colors = ['red', 'yellow', 'blue', 'green']

# Create subplots
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(14, 9), dpi=100)

# Grouped Bar Chart
width = 0.2  # Width of the bars
x = range(len(sectors))  # X locations for the groups

for i, attack_type in enumerate(attack_types):
    ax1.bar([p + width*i for p in x], df.loc[attack_type], width, label=attack_type, color=colors[i])

ax1.set_xlabel('Top five industry sectors')
ax1.set_ylabel('Number of attacks')
ax1.set_title('Malicious or criminal attack breaches - Top 5 sectors')
ax1.set_xticks([p + 1.5 * width for p in x])
ax1.set_xticklabels(sectors, rotation=90)
ax1.legend(title='Attack types')

# Show values on top of bars
for bars in ax1.containers:
    ax1.bar_label(bars)

# Stacked Bar Chart
bottom = [0] * len(sectors)
for i, attack_type in enumerate(attack_types):
    ax2.bar(sectors, df.loc[attack_type], bottom=bottom, label=attack_type, color=colors[i])
    bottom = [i+j for i, j in zip(bottom, df.loc[attack_type])]

ax2.set_xlabel('Top five industry sectors')
ax2.set_ylabel('Number of attacks')
ax2.set_title('Malicious or criminal attack breaches - Top 5 sectors')
ax2.set_xticklabels(sectors, rotation=90)
ax2.legend(title='Attack types')

plt.tight_layout()
plt.show()
```

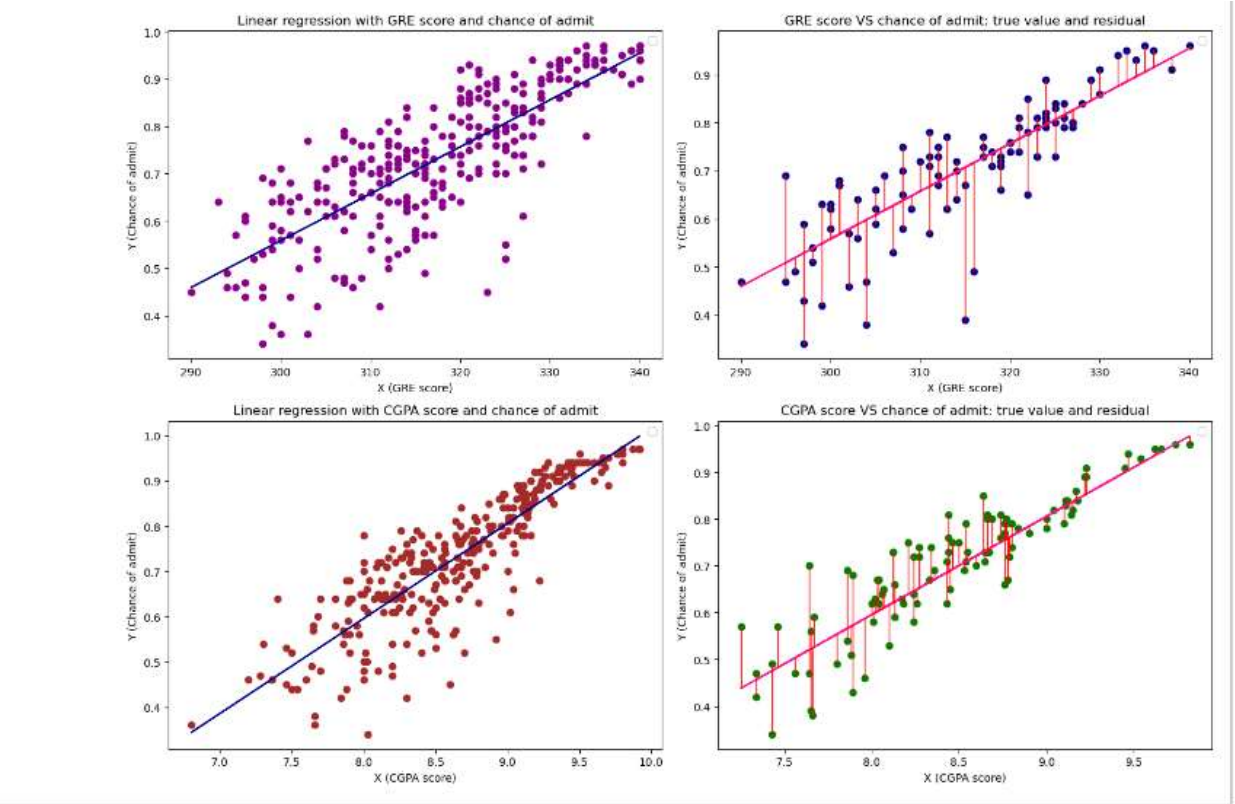

Malicious or criminal attack breaches – Top 5 sectors
Attack types
Cyber incident
Social engineering / impersonation
Rogue employee / insider threat
Theft of paperwork or data storage device
Number of attacks
Health service providers
Finance (incl. superannuation)
Insurance
Retail
Australian Government
Top five industry sectors
Malicious or criminal attack breaches – Top 5 sectors
Attack types
Cyber incident
Social engineering / impersonation
Rogue employee / insider threat
Theft of paperwork or data storage device
Number of attacks
Health service providers
Finance (incl. superannuation)
Insurance
Retail
Australian Government
Top five industry sectors

Credit Task 5.2C: Linear Regression

The dataset contains several parameters which are considered important during the application for Masters Programs. The parameters included are:

1. GRE Scores (out of 340)
2. TOEFL Scores (out of 120)
3. University Rating (out of 5)
4. Statement of Purpose and Letter of Recommendation Strength (out of 5)
5. Undergraduate GPA (out of 10)
6. Research Experience (either 0 or 1)
7. Chance of Admit (ranging from 0 to 1)

```
# Load the dataset
df = pd.read_csv('admission_predict.csv')

# Split the dataset into training and testing sets
train_data = df[:300]
test_data = df[300:]

# Features and target variables for training
X_train_GRE = train_data[['GRE Score']]
X_train_CGPA = train_data[['CGPA']]
y_train = train_data['Chance of Admit']

# Features for testing
X_test_GRE = test_data[['GRE Score']]
X_test_CGPA = test_data[['CGPA']]
y_test = test_data['Chance of Admit']

# Linear regression model for GRE Score
model_GRE = LinearRegression()
model_GRE.fit(X_train_GRE, y_train)

# Linear regression model for CGPA
model_CGPA = LinearRegression()
model_CGPA.fit(X_train_CGPA, y_train)

# Predictions
y_pred_train_GRE = model_GRE.predict(X_train_GRE)
y_pred_train_CGPA = model_CGPA.predict(X_train_CGPA)
y_pred_test_GRE = model_GRE.predict(X_test_GRE)
y_pred_test_CGPA = model_CGPA.predict(X_test_CGPA)

# Visualization
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(14, 10), dpi=100)

# Plot for GRE Score (Train)
ax[0, 0].scatter(X_train_GRE, y_train, color='#8B008B')
ax[0, 0].plot(X_train_GRE, y_pred_train_GRE, color='#00008B')
ax[0, 0].set_title('Linear regression with GRE score and chance of admit')
ax[0, 0].set_xlabel('X (GRE score)')
ax[0, 0].set_ylabel('Y (Chance of admit)')
ax[0, 0].legend()

# Plot for GRE Score (Test)
ax[0, 1].scatter(X_test_GRE, y_test, color='#00008B')
ax[0, 1].plot(X_test_GRE, y_pred_test_GRE, color='#FF1493')
# Plot residuals as straight lines
for i in range(len(X_test_GRE)):
    ax[0, 1].plot([X_test_GRE.iloc[i], X_test_GRE.iloc[i]], [y_test.iloc[i], y_pred_test_GRE[i]], color='red', linestyle='-', li
ax[0, 1].set_title('GRE score VS chance of admit: true value and residual')
ax[0, 1].set_xlabel('X (GRE score)')
ax[0, 1].set_ylabel('Y (Chance of admit)')
ax[0, 1].legend()

# Plot for CGPA (Train)
ax[1, 0].scatter(X_train_CGPA, y_train, color='#A52A2A')
ax[1, 0].plot(X_train_CGPA, y_pred_train_CGPA, color='#00008B')
ax[1, 0].set_title('Linear regression with CGPA score and chance of admit')
ax[1, 0].set_xlabel('X (CGPA score)')
ax[1, 0].set_ylabel('Y (Chance of admit)')
ax[1, 0].legend()

# Plot for CGPA (Test)
ax[1, 1].scatter(X_test_CGPA, y_test, color='#008000')
ax[1, 1].plot(X_test_CGPA, y_pred_test_CGPA, color='#FF1493')
# Plot residuals as straight lines
for i in range(len(X_test_CGPA)):
    ax[1, 1].plot([X_test_CGPA.iloc[i], X_test_CGPA.iloc[i]], [y_test.iloc[i], y_pred_test_CGPA[i]], color='red', linestyle='-',
ax[1, 1].set_title('CGPA score VS chance of admit: true value and residual')
ax[1, 1].set_xlabel('X (CGPA score)')
ax[1, 1].set_ylabel('Y (Chance of admit)')
ax[1, 1].legend()

plt.tight_layout()
plt.show()
```



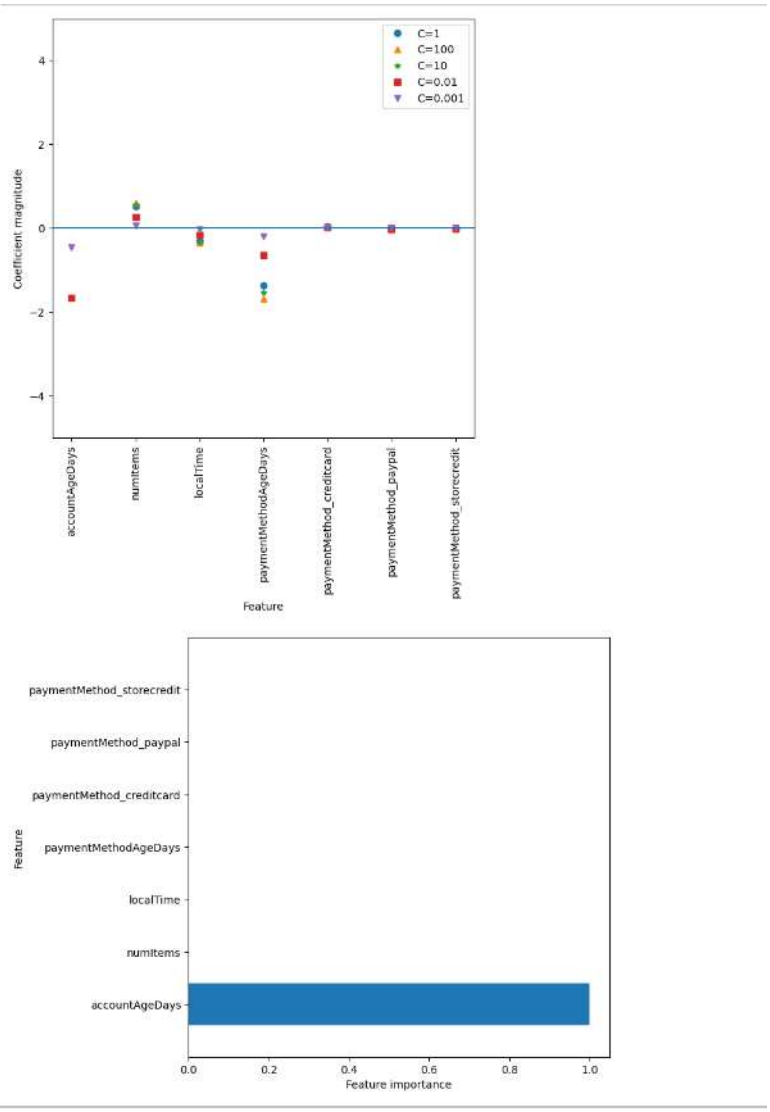
Utilizing the "Admission Predict" dataset, the algorithm applies linear regression analysis to forecast the "Chance of Admit" according to the "CGPA" and "GRE Score." Pandas is used to import the dataset first, after which it is divided into training (the first 300 rows) and testing (the remaining rows) groups. Next, using "GRE Score" and "CGPA" as features, distinct Linear Regression models are developed.

Training and testing data predictions are generated with the taught models. The findings are then plotted on distinct subplots for the training and testing sets for the GRE Score and CGPA, showing actual values against anticipated using matplotlib.

Relative to the anticipated values, or residuals, are likewise displayed as red lines on the test set plots. The charts illustrate the fit of the models and the size of prediction errors, offering insights into how effectively the models predict the "Chance of Admit" based on the "GRE Score" and "CGPA."

Distinction Task 6.2D: LogisticRegression and Decision tree

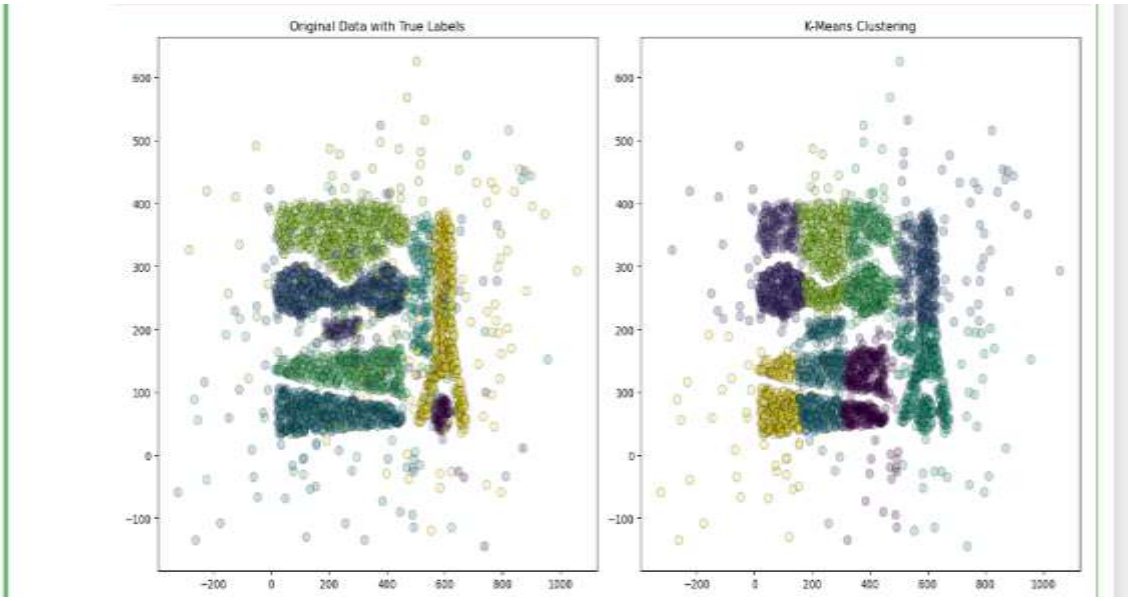
```
]: #importing libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

# Load the dataset
data = pd.read_csv('payment_fraud.csv')

# Convert categorical variable into dummy variables
data = pd.get_dummies(data, columns=['paymentMethod'])

# Splitting the data using train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    data.drop('label', axis=1), data['label'], test_size=0.33, stratify=data['label'], random_state=17
)

# Initialize Logistic Regression models with different C parameters
logreg1 = LogisticRegression(C=1, max_iter=10000, random_state=42)
logreg100 = LogisticRegression(C=100, max_iter=10000, random_state=42)
logreg10 = LogisticRegression(C=10, max_iter=10000, random_state=42)
logreg001 = LogisticRegression(C=0.01, max_iter=10000, random_state=42)
logreg0001 = LogisticRegression(C=0.001, max_iter=10000, random_state=42)

# Fit the models
logreg1.fit(X_train, y_train)
logreg100.fit(X_train, y_train)
logreg10.fit(X_train, y_train)
logreg001.fit(X_train, y_train)
logreg0001.fit(X_train, y_train)

# Plotting the graph for coefficients
plt.subplots(figsize=(7, 7), dpi=100)
plt.plot(logreg1.coef_.T, 'o', label="C=1")
plt.plot(logreg100.coef_.T, '^', label="C=100")
plt.plot(logreg10.coef_.T, '*', label="C=10")
plt.plot(logreg001.coef_.T, 's', label="C=0.01")
plt.plot(logreg0001.coef_.T, 'v', label="C=0.001")
plt.xticks(range(X_train.shape[1]), X_train.columns, rotation=90)
xlims = plt.xlim()
plt.hlines(0, xlims[0], xlims[1])
plt.xlim(xlims)
plt.ylim(-5, 5)
plt.xlabel("Feature")
plt.ylabel("Coefficient magnitude")
plt.legend()
plt.show()

# Decision tree learning algorithm
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train)

# Plotting the decision tree for feature importance
plt.subplots(figsize=(7, 7), dpi=100)
n_features = X_train.shape[1]
plt.barh(np.arange(n_features), tree.feature_importances_, align='center')
plt.yticks(np.arange(n_features), X_train.columns)
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.ylim(-1, n_features)
plt.show()
```

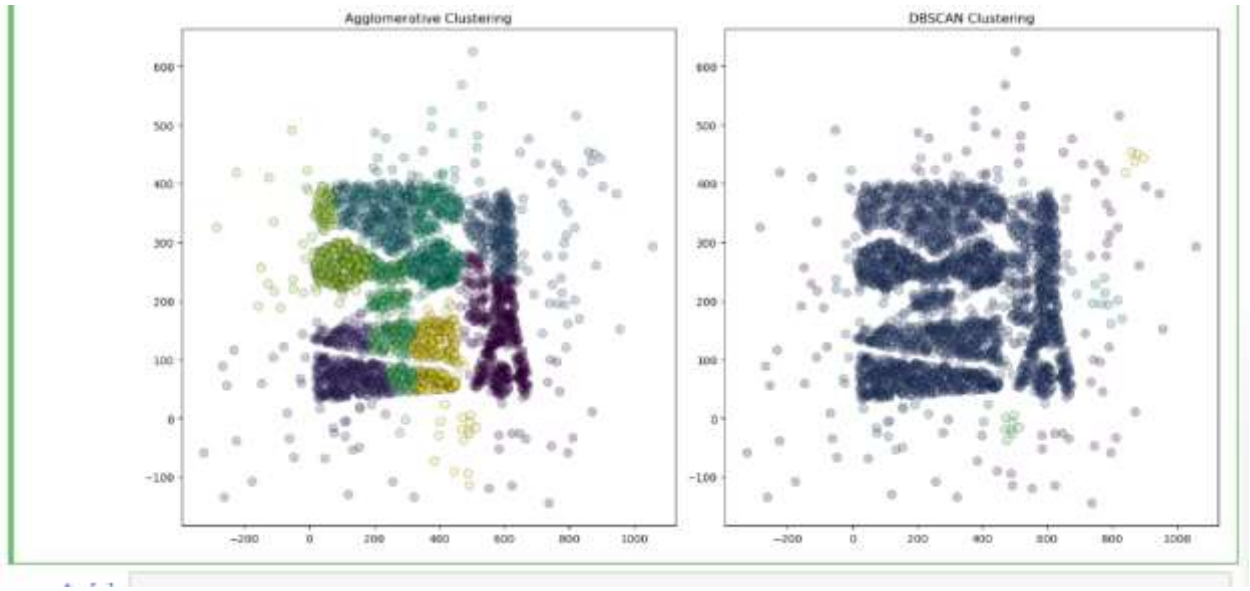


This Python script uses Decision Tree and Logistic Regression classifiers on a payment fraud dataset. The data is first loaded, preprocessed, and divided into training and test sets. Categorical variables are then changed into dummy variables. Plotting the coefficients of trained logistic regression models with varying regularization strengths (C values) allows one to see the impact of features. After training a Decision Tree classifier, the most significant characteristics for fraud prediction are shown in a bar plot showing feature importance. This script illustrates the importance of features in classification as well as model behavior.

8.2D
Comparing unsupervised clustering algorithms

In [7]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.preprocessing import StandardScaler

# Load the dataset
df = pd.read_csv('Complex8_N15.csv')

# Extract the features and the true labels
X = df[['V1', 'V2']].values
y_true = df['V3'].values

# Normalize the dataset for DBSCAN
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Create subplots for comparison
fig, axs = plt.subplots(2, 2, figsize=(14, 14), dpi=100)

# Plot original data with true labels
axs[0, 0].scatter(X[:, 0], X[:, 1], c=y_true, cmap='viridis', alpha=0.25, s=60, linewidths=1, edgecolors='black')
axs[0, 0].set_title('Original Data with True Labels')

# K-Means clustering
kmeans = KMeans(n_clusters=8, random_state=42)
y_kmeans = kmeans.fit_predict(X)
axs[0, 1].scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis', alpha=0.25, s=60, linewidths=1, edgecolors='black')
axs[0, 1].set_title('K-Means Clustering')

# Agglomerative Clustering
agglo = AgglomerativeClustering(n_clusters=8)
y_agglo = agglo.fit_predict(X)
axs[1, 0].scatter(X[:, 0], X[:, 1], c=y_agglo, cmap='viridis', alpha=0.25, s=60, linewidths=1, edgecolors='black')
axs[1, 0].set_title('Agglomerative Clustering')

# DBSCAN Clustering
dbscan = DBSCAN(eps=0.3, min_samples=5)
y_dbscan = dbscan.fit_predict(X_scaled)
axs[1, 1].scatter(X[:, 0], X[:, 1], c=y_dbscan, cmap='viridis', alpha=0.25, s=60, linewidths=1, edgecolors='black')
axs[1, 1].set_title('DBSCAN Clustering')

# Adjust layout and show plot
plt.tight_layout()
plt.show()
```





Using a provided 2-dimensional dataset (Complex8_N15.csv), this Python script compares three unsupervised clustering algorithms: K-Means, Agglomerative Clustering, and DBSCAN. The panda's package is used to import the dataset, and then the pertinent features (V1 and V2) and their matching true labels (V3) are retrieved. To guarantee that every feature contributes equally to the clustering process, the dataset is normalised using the StandardScaler before the clustering algorithms are applied. This is crucial for DBSCAN because it relies heavily on distance-based computations.

In order to visually compare the original data distribution and the clustering outcomes from each technique, the script generates a collection of subplots. The first subplot uses a scatter plot to display the actual data with true labels, with different colours denoting different classes (V3). The K-Means clustering algorithm is used in the following subplot, with 8 clusters specified (n_clusters=8). To see how K-Means divides the data points into clusters, the outcomes are plotted.

Similar to this, eight clusters are used for agglomerative clustering, and the outcomes are shown in a different subplot. Data points are grouped using the hierarchical Agglomerative Clustering methodology according to how similar they are; this is represented in the clustering result.

Lastly, the normalised data (X_scaled) is subjected to the DBSCAN algorithm, which has two parameters: min_samples=5 (the minimum number of samples in a neighbourhood for a point to be considered a core point) and eps=0.3 (the maximum distance between two samples for one to be considered as in the neighbourhood of the other). The final subplot shows the resulting clusters.

The script makes it possible to compare the clustering results side by side with the original labelled data by setting up all of the plots in a 2x2 grid format. This makes it simpler to see the similarities and differences amongst the three clustering algorithms. The visual comparison is guaranteed to be understandable and intuitive through the use of standard plot styles (colour maps, marker sizes, and transparency levels). Plt.tight_layout () is used to improve readability of the layout, while plt.show () is used to display the plots.

Distinction Task 10.1D: Model evaluation metrics

- **An analysis of a fraud detection model utilizing decision trees and K-Nearest Neighbors (KNN)**
  Detecting fraudulent credit card transactions from a data set is the goal of this Python-based machine learning project, which makes use of the supervised learning models, KNN and Decision Trees. Class labels that aid in determining whether a transaction is fraudulent and normalized transacted value are among the variables in the dataset, along with raw values for dependent and independent variables. This study does a good job of mitigating data distortion, especially when it comes to a low percentage of fraudulent transactions.

- **Data Preprocessing**
  The dataset's Amount feature set is chosen from the previously downloaded CSV file. The StandardScaler from sklearn is then used to normalize this feature set. Finally, the features, Time and Amount, are removed. The data has a significant class imbalance because the percentage of fraudulent transactions makes up a very small component of the data. Under sampling, which lowers the percentage of honest transactions to match the proportion of fraudulent ones, is used to solve the issue. Consequently, a balanced under sampled dataset is produced, and it is further split into training and testing data sets, which have a 7:3 ratio, respectively.

- **K-Nearest neighbors (KNN)**
  Grid Search CV is used to train the KNN model on an undersampled dataset in order to improve the hyper parameter n_neighbors. The value of n_neighbors is tested between 1 and 5. The goal of cross-validation is to estimate the optimal value of "n_neighbors," and the recall is the next parameter used to select this value. The recall measure is highlighted because false negatives, or missed fraudulent cases, are among the most important components in fraud analysis.

  The test set's confusion matrix is displayed, and the recall value is computed, to assess the model's performance. The recall score for the program's capacity to spot fraudulent transactions serves as evidence of this. Ultimately, the complete dataset is run through the KNN model to equitably evaluate the algorithm's performance on the undersampled data.

- **Threshold Tuning**
  Better model outcomes are achieved by fine-tuning the category determination thresholds. The probabilities calculated by the KNN classifier for different thresholds (between 0. 1 and 0. 9) are computed at each stage of the procedure, along with confusion matrices and recall metrics. The goal of this stage is to enable finding a good balance by determining the optimal recall level and precision level.

- **Decision Tree Classifier**
  In addition to KNN classifiers, Decision Tree classifiers are also implemented. Tuning the Decision Tree's characteristic max_depth, whose values vary from 3 to 7, is what cross-validation entails. Once more, the model is tested using cross-validation, with recall being the primary focus this time. KNN is the format utilized, and accuracy and recall are given for both the full and under sample data sets. One can determine whether model is more effective at identifying fraudulent transactions by comparing the Decision Tree and KNN models in the following comparison.

- **Model Assessment**
  Therefore, while evaluating both models, the recall and confusion matrix outcomes are used, with a focus on the quantity of false negatives. The ability to see how the models' accuracy and recall are swapped at various thresholds is one of the benefits of the charting method, which aids in the models' further improvement.

- **Conclusion**
  This paper describes how to detect fraud transactions in unbalanced datasets using KNN and Decision Tree classifiers, based on the research. One of the key evaluation metrics is the confusion matrix, and recall is taken into account while adjusting the models. To enhance fraud detection, the models can be further refined by employing under sampling to reduce data imbalances and adjusting the bounds of classification.

Cyber Security Analytics
Task 6.3HD

This research assesses the performance of four classification algorithms using the Spambase dataset: Random Forest, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Logistic Regression. The dataset's goal is to determine whether an email is spam or not. It consists of 4601 examples with 57 attributes.

```
In [2]: import pandas as pd
        import numpy as np
        from sklearn.model_selection import train_test_split
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.linear_model import LogisticRegression
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.svm import SVC
        from sklearn.preprocessing import StandardScaler
        from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
        import matplotlib.pyplot as plt
        import seaborn as sns

        # Load dataset
        df = pd.read_csv("spambase.data", header=None)
        X = df.iloc[:, :-1]
        y = df.iloc[:, -1]

        # Ensure X is a NumPy array and contiguous in memory
        X = np.ascontiguousarray(X)

        # Split dataset into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

        # Standardize data for Logistic Regression and SVM
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Function to plot confusion matrix
        def plot_conf_matrix(y_test, y_pred, model_name):
            cm = confusion_matrix(y_test, y_pred)
            plt.figure(figsize=(6, 5))
            sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
            plt.xlabel('Predicted')
            plt.ylabel('Actual')
            plt.title(f'{model_name} Confusion Matrix')
            plt.show()

        # 1. K-Nearest Neighbors (KNN)
        knn = KNeighborsClassifier(n_neighbors=5)
        knn.fit(X_train, y_train)
        y_pred_knn = knn.predict(X_test)
        print("KNN - Training Accuracy: {:.3f}".format(knn.score(X_train, y_train)))
        print("KNN - Test Accuracy: {:.3f}".format(accuracy_score(y_test, y_pred_knn)))

        # Plot Confusion Matrix for KNN
        plot_conf_matrix(y_test, y_pred_knn, "KNN")
```

```
        # 2. Logistic Regression
        log_reg = LogisticRegression(solver='lbfgs', max_iter=5000, C=1)
        log_reg.fit(X_train_scaled, y_train)
        y_pred_log_reg = log_reg.predict(X_test_scaled)
        print("Logistic Regression - Training Accuracy: {:.3f}".format(log_reg.score(X_train_scaled, y_train)))
        print("Logistic Regression - Test Accuracy: {:.3f}".format(accuracy_score(y_test, y_pred_log_reg)))

        # Plot Confusion Matrix for Logistic Regression
        plot_conf_matrix(y_test, y_pred_log_reg, "Logistic Regression")

        # Plot coefficients for Logistic Regression
        plt.figure(figsize=(10, 6))
        plt.plot(log_reg.coef_.T, 'o', label="C=1")
        plt.xticks(range(X_train.shape[1]), range(X_train.shape[1]), rotation=90)
        plt.xlabel("Feature index")
        plt.ylabel("Coefficient magnitude")
        plt.title("Logistic Regression Coefficients")
        plt.legend()
        plt.show()

        # 3. Random Forest
        forest = RandomForestClassifier(n_estimators=100, random_state=42)
        forest.fit(X_train, y_train)
        y_pred_forest = forest.predict(X_test)
        print("Random Forest - Training Accuracy: {:.3f}".format(forest.score(X_train, y_train)))
        print("Random Forest - Test Accuracy: {:.3f}".format(accuracy_score(y_test, y_pred_forest)))

        # Plot Confusion Matrix for Random Forest
        plot_conf_matrix(y_test, y_pred_forest, "Random Forest")

        # Plot Feature Importances for Random Forest
        plt.figure(figsize=(10, 6))
        plt.bar(range(X_train.shape[1]), forest.feature_importances_, align='center')
        plt.xticks(range(X_train.shape[1]), range(X_train.shape[1]))
        plt.xlabel("Feature Importance")
        plt.ylabel("Feature Index")
        plt.title("Random Forest Feature Importances")
        plt.show()

        # 4. Support Vector Machines (SVM)
        svm_model = SVC(kernel='linear', C=1, random_state=42)
        svm_model.fit(X_train_scaled, y_train)
        y_pred_svm = svm_model.predict(X_test_scaled)
        print("SVM - Training Accuracy: {:.3f}".format(svm_model.score(X_train_scaled, y_train)))
        print("SVM - Test Accuracy: {:.3f}".format(accuracy_score(y_test, y_pred_svm)))

        # Plot Confusion Matrix for SVM
        plot_conf_matrix(y_test, y_pred_svm, "SVM")
```

```
        # Plot Confusion Matrix for SVM
        plot_conf_matrix(y_test, y_pred_svm, "SVM")

        # Plot Classification Reports
        classification_reports = {
            "KNN": classification_report(y_test, y_pred_knn, output_dict=True),
            "Logistic Regression": classification_report(y_test, y_pred_log_reg, output_dict=True),
            "Random Forest": classification_report(y_test, y_pred_forest, output_dict=True),
            "SVM": classification_report(y_test, y_pred_svm, output_dict=True)
        }

        # Visualize classification reports using heatmaps
        for model, report in classification_reports.items():
            plt.figure(figsize=(10, 6))
            sns.heatmap(pd.DataFrame(report).iloc[:-1, :].T, annot=True)
            plt.title(f"{model} Classification Report")
            plt.show()
```

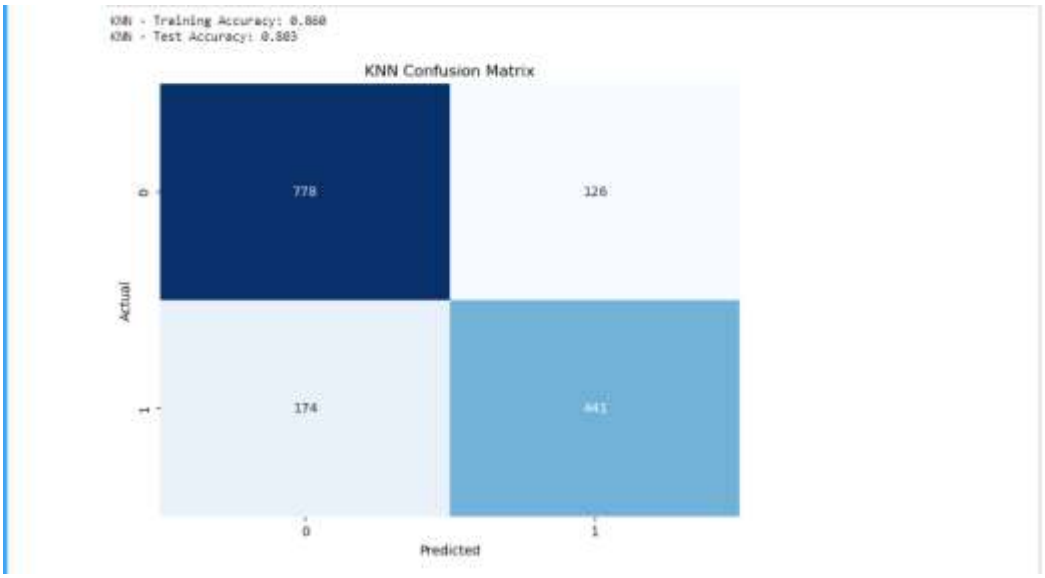
Methodology

Model Results

Model	Training Accuracy	Test Accuracy
K-Nearest Neighbors (KNN)	0.860	0.803
Logistic Regression	0.937	0.910
Random Forest	1.000	0.948
Support Vector Machine	0.939	0.914

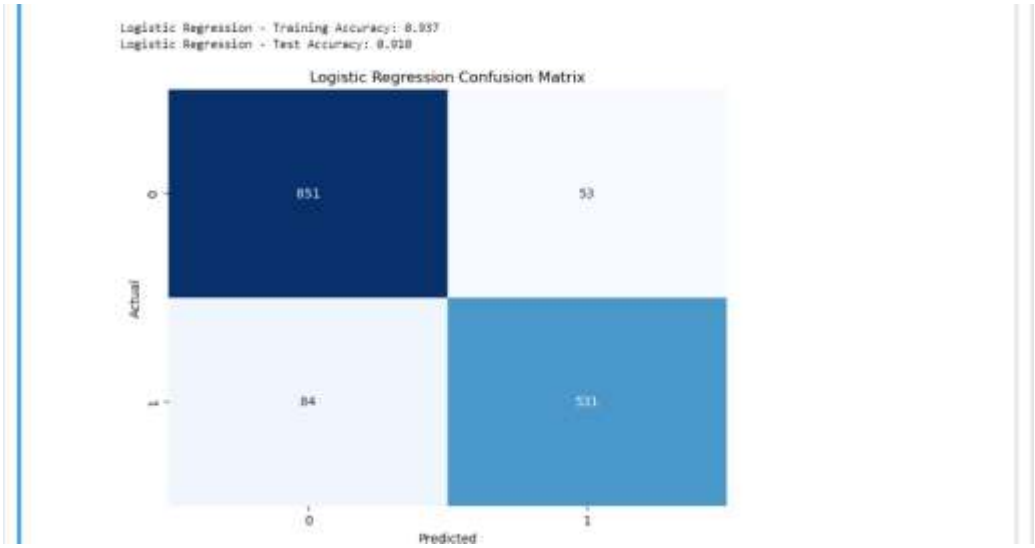
K-Nearest Neighbors (KNN)

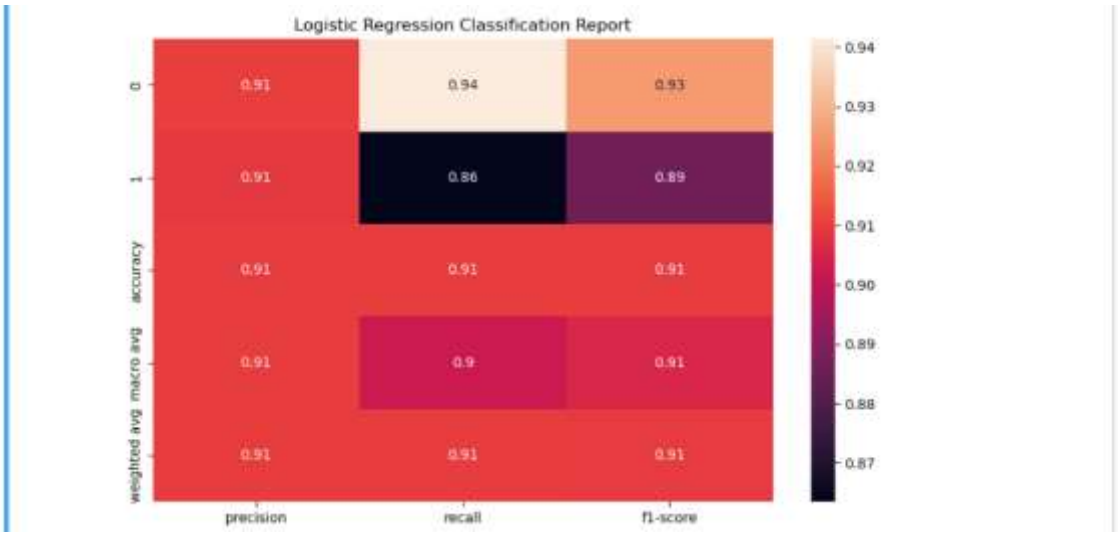
A training accuracy of 86.0% and 80. Despite having only 3% accuracy in testing, the KNN model was improved by competition. Although KNN's accuracy is commendable, its lower training accuracy suggests that it may have had trouble choosing the underlying the data set's patterns. The model does not produce great accuracy even in the test, and it can be concluded that the model's inability to generalize may be caused by the excessive model's sensitivity to duplicate information and noise in the training set of data. How well it discriminates between emails that are spam and those that are not is demonstrated by the confusion matrix of the KNN. The higher number of misclassifications the model showed compared to the other models illustrated its lower accuracy.



Logistic Regression

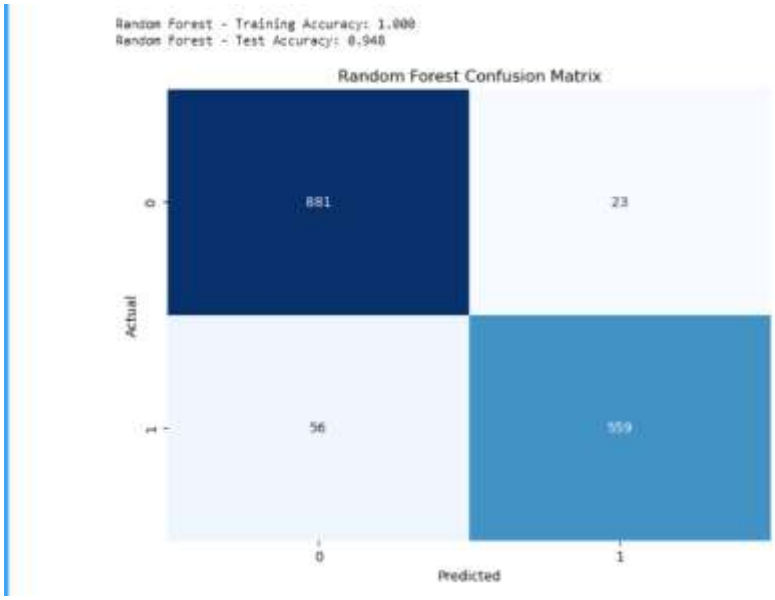
A training accuracy of 93 has been achieved by the use of logistic regression after the illness. In all of the trials, the model attains a 7% accuracy rate and a 91.0 % test accuracy. Additionally, in this instance, the model exhibited excellent training-test generalization, which means that the best patterns were discovered from the training set without discovering trivia from it as well. The Logistic Regression model utilized for categorizing the detected emails demonstrated a strong ability to distinguish between emails that were spam and those that weren't, based on the accuracy parameters. The model's coefficients can be analyzed to ascertain the elements' relevance. That specify the model. A high proportion of correctly classified data and a low percentage of incorrectly classified data are displayed in the confusion matrix for logistic regression, indicating decent performance. Effectively, the model separates the two classes.

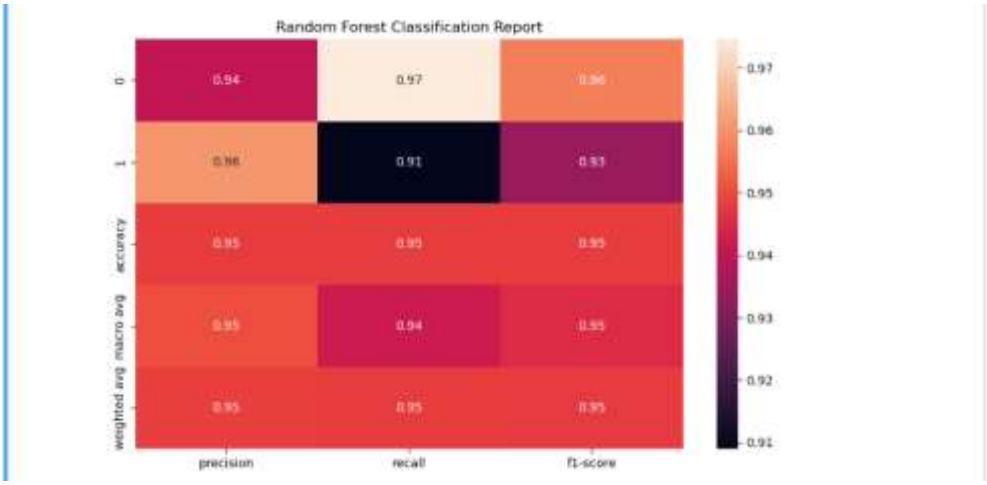




Random Forest

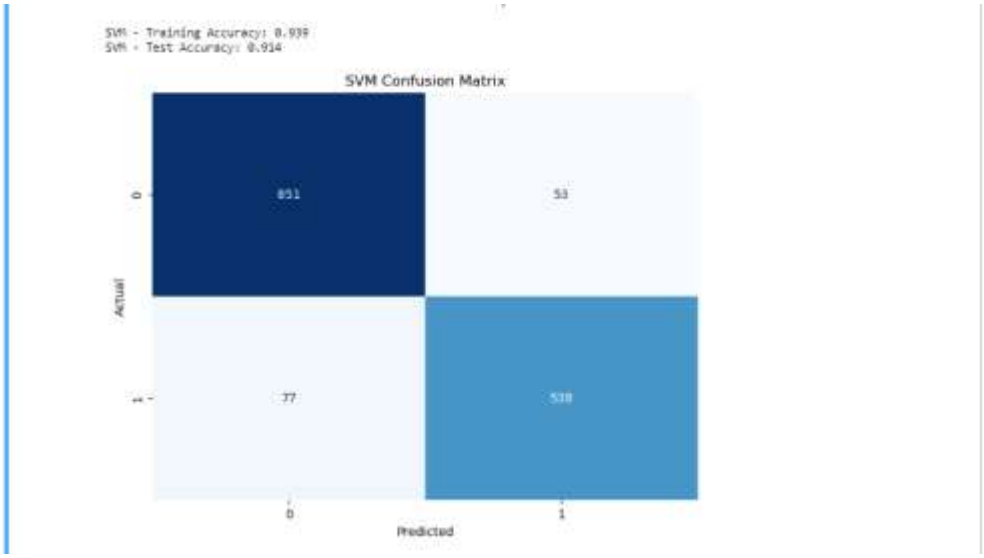
With a test accuracy of 94.8% and a training accuracy of 100.0%, RF is the model with the best accuracy. When a model is considered accurate in its training, it has mastered every aspect of the input data and produced flawless results on the training set. The Random Forest model is the best of all the models shown in this comparison, and it's extremely high test accuracy indicates that they have excellent generalization. The model's feature significance outputs display the most significant characteristics on the dataset that are indicative of spam. The confusion matrix for Random Forest shows nearly perfect performance with few errors. This provides even more support for the model's top accuracy scores.

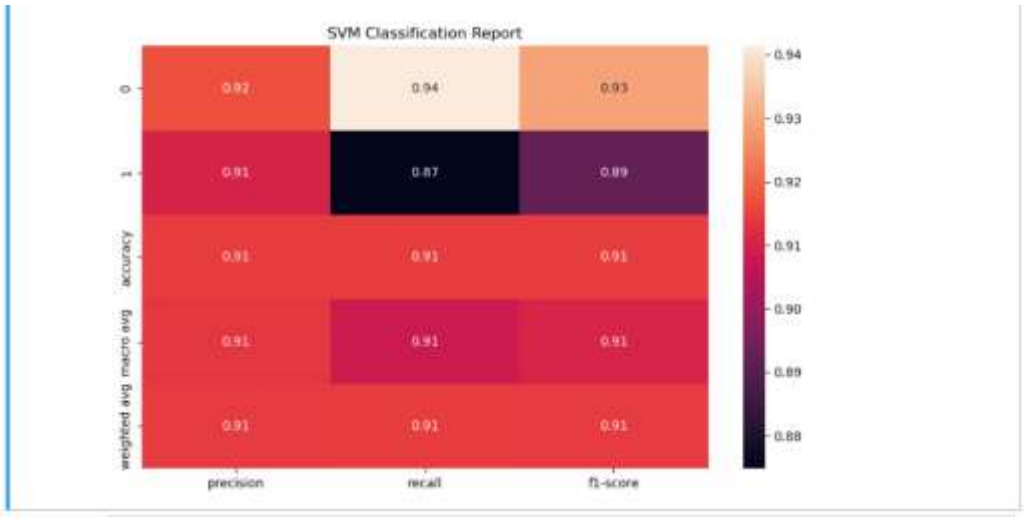




Support Vector Machine (SVM)

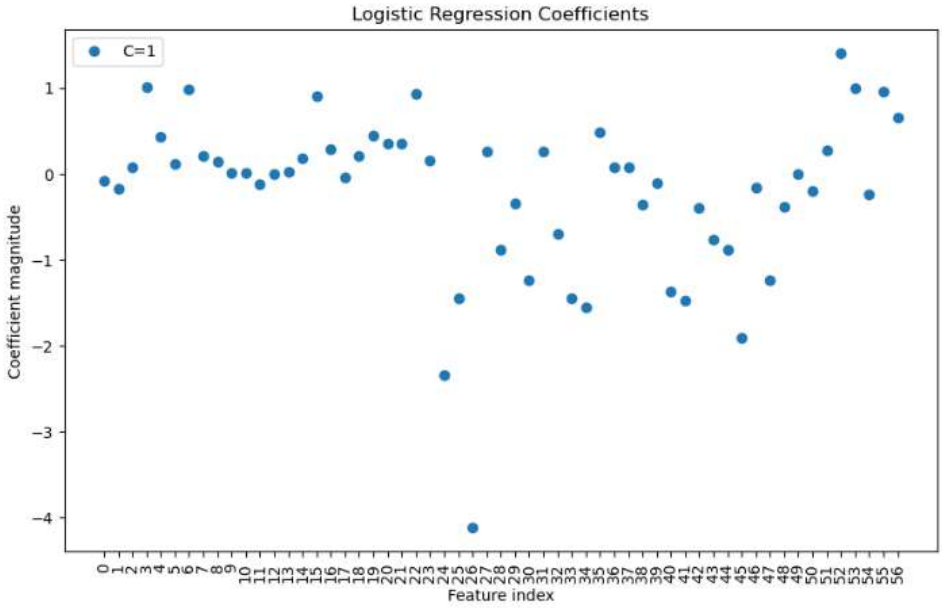
It was discovered that the SVM model has a 91% test accuracy and a 93% training accuracy. This model did a fantastic job of both acting as a good model for unknown data and imitating the patterns of the training set. According to the test accuracy results, SVM was actually marginally more accurate than Logistic Regression in correctly identifying spam emails. If the model's decision boundary were displayed, it would show the type of region in the feature space where spam and no spam are found. A robust and balanced performance is demonstrated by the SVM confusion matrix, which has very few wrong classifications. It is similar to the results of Random Forest.





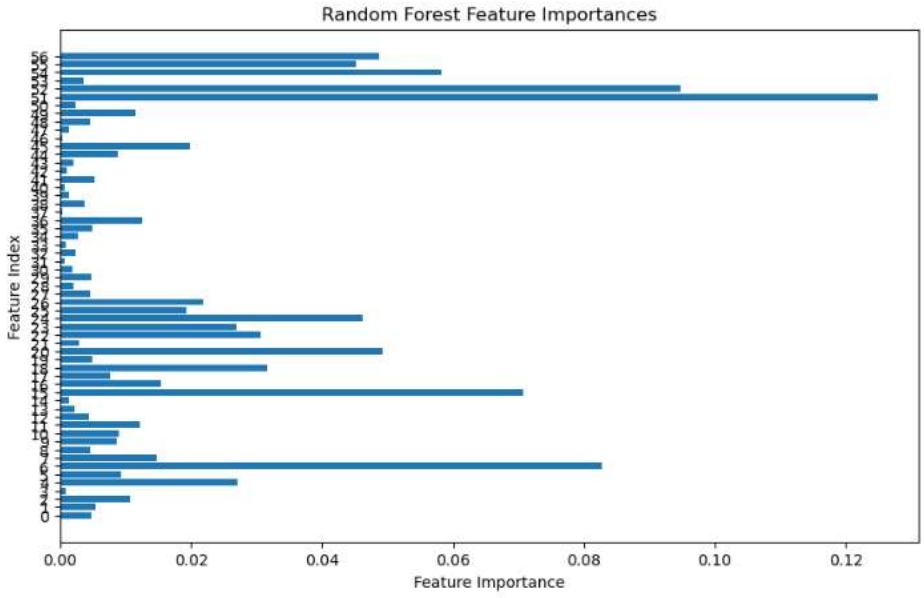
Logistic Regression Coefficients

To understand the findings and ascertain the contribution of each variable in identifying the spam, a plot of the Logistic Regression model's coefficients was made. This means that the best indicators of whether an email may be spam are those features with high coefficients. This plot is utilized in the model's interpretation to ascertain the importance of the different attributes.



Random Forest Feature Importance

The random forest model's feature imports were to be displayed using a created bar chart. This figure helps to illustrate the properties of the dataset and aids in determining the patterns of spam classification by highlighting the attributes that the model relied on most heavily during the prediction phase.



Report: Task 10.2HD

- Thus, the goal of this work is to enhance the model by evaluating and utilizing a variety of machine learning techniques on an unbalanced credit card dataset. These include of methods like features extraction, majority class undersampling, and minority class oversampling. Preliminary boosting methods including LightGBM, CatBoost, and XGBoost are included for further model improvements. The main evaluation metric in this instance was the same Area Under the Receiver Operating Characteristic Curve, or AUC-ROC, which is frequently used in classification issues, especially in imbalanced data sets like the one in this instance. Other measures were also calculated for information and classification pattern analysis.

- ✓ Since the first step in this approach was to undertake exploratory data analysis, or EDA, I started by attempting to comprehend the data as it is with the ultimate goal of identifying different factors that might affect the model's performance. I used undersampling and oversampling techniques because of a serious class imbalance issue, in which the number of fraudulent transactions is far lower than the total amount of data. In order to compare with other models, the baseline model was initially assessed using the Random Forest classifier.

- ✓ Although we might have thought about utilizing Principal Component Analysis (PCA), this was immediately rejected as the Therefore, there wasn't much to gain from dimensionality reduction in the data set. Rather, my emphasis was on employing SMOTE (Synthetic Minority Over-sampling Technique) to equalize the data set and identify instances of fraud related to the minority class.

- ✓ I created a synthetic data set with 1128 samples, 10 attributes, a trained Random Forest classifier, and a goal variable: binary classes for the experiment. The precision of the model was not as significant as the correctness of the probabilistic forecasts. At first, I saw that even if the model has a high classification accuracy, there are certain problems with non-varying probability estimates. Thus, I used the sigmoid and isotonic calibration procedures in an effort to improve the probability calibration.

- ✓ To be more specific, I divided the provided dataset into test, calibration, and training sets in order to carry out the problem-solving procedure. I then followed the training set's instructions to train the Random Forest model, using the calibration set to modify the probability estimates. After training the models, I calibrated the models using two calibrations techniques utilizing classifiers that are pre-fitted in scikit-learn’s CalibratedClassifierCV class. The decision was made to proceed with probability calibration instead of repeating the models that produced these probabilities because probability calibration was the primary focus.

The Output screenshot of 10.2HD ipynb Task

1.2 Importing and fixing the csv file

Based off 10.1D, let's normalise the Amount column and drop the Time column before continuing.

[3]:
```
# Load the dataset
df = pd.read_csv('creditcard.csv')

# Rescale the 'Amount' column using StandardScaler and create a new column 'normAmount'
df['normAmount'] = StandardScaler().fit_transform(df['Amount'].values.reshape(-1, 1))

# Drop the 'Time' and original 'Amount' columns as they are not required for further analysis
df = df.drop(['Time', 'Amount'], axis=1)

# Display the first few rows to ensure the changes are applied correctly
df.head()
```

[3]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V24	V25
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.066928	0.128539
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.689281	-0.327642
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010

5 rows × 30 columns

jupyter 10.2HD sample Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

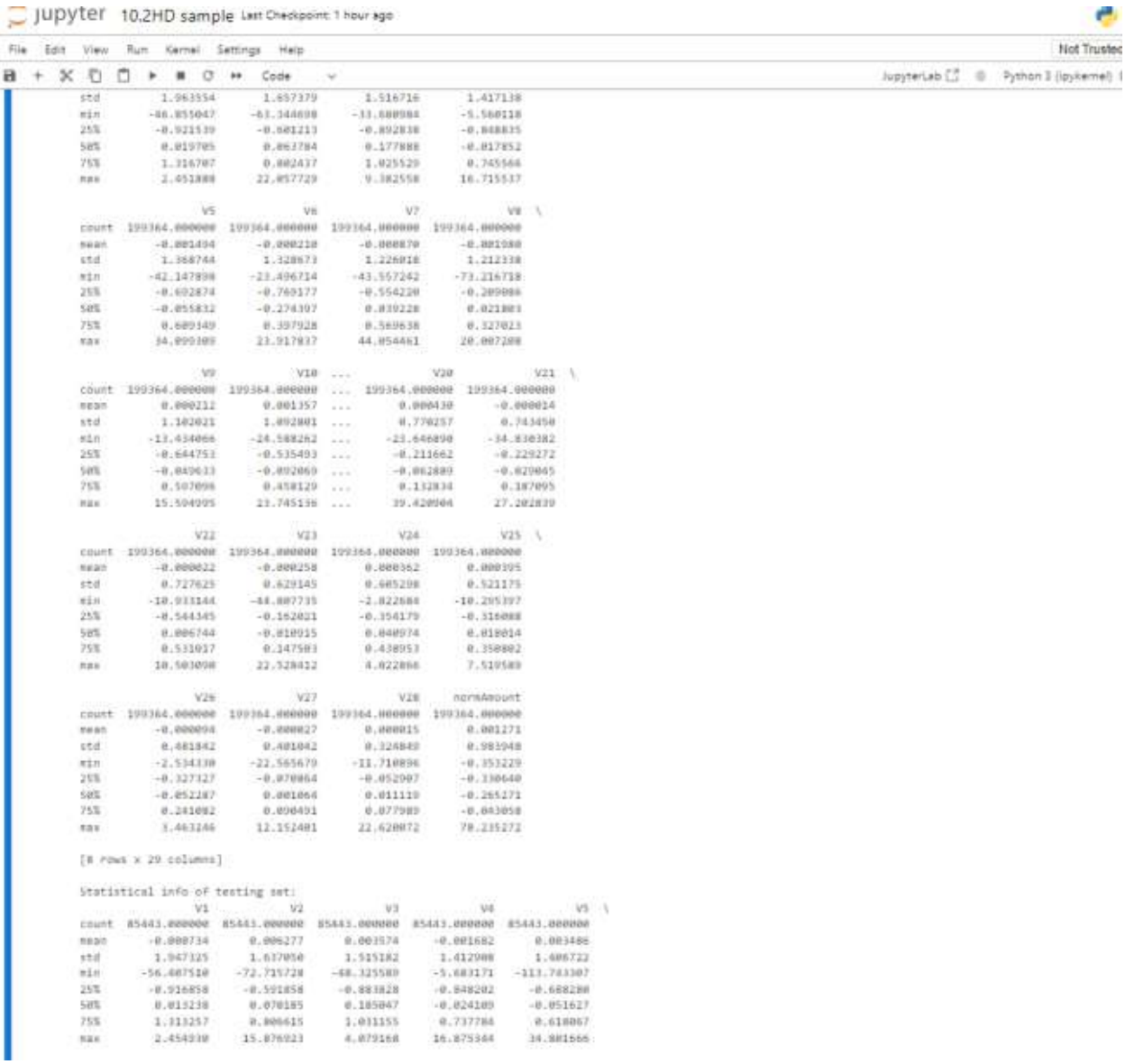
Requirements: Split the dateset as suggested (e.g. at ratio 0.3) with random_state set. Statistical info about the split datasets expected, such as shape, sample count.

1.3 Data

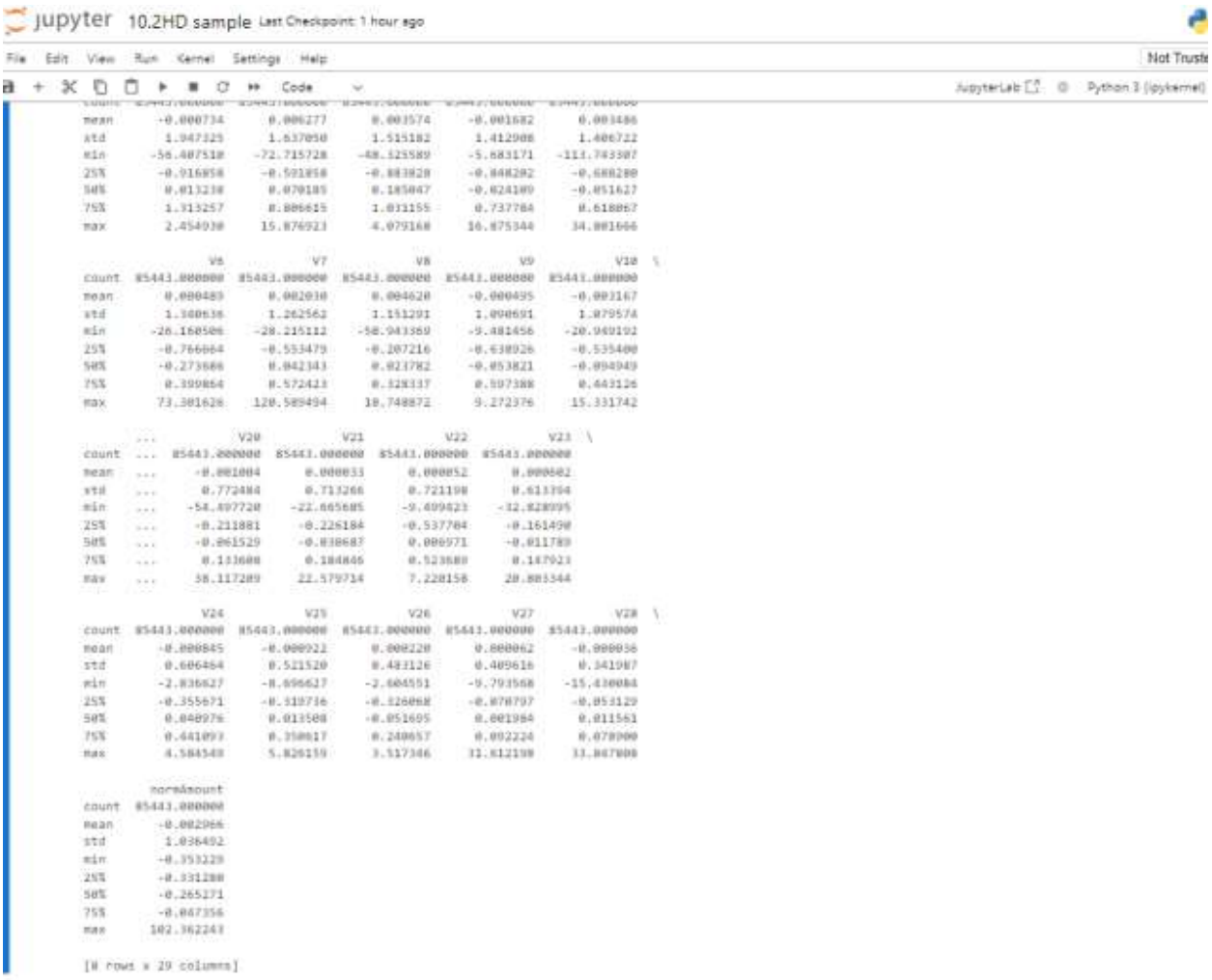
Let's split the dataset into training and testing sets. Also check their shapes.

[5]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Assuming df is your DataFrame already loaded with data

# Get data points and labels
X = df.iloc[:, df.columns != 'Class']
y = df.iloc[:, df.columns == 'Class'].values.ravel()  # Flatten y if it's a 2D array

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Reset indices to avoid issues with non-existent indices
X_train = X_train.reset_index(drop=True)
y_train = pd.Series(y_train).reset_index(drop=True)  # Convert to Series for consistency
X_test = X_test.reset_index(drop=True)
y_test = pd.Series(y_test).reset_index(drop=True)  # Convert to Series for consistency

# View shapes
print(f'X_train\'s shape : {X_train.shape}')
print(f'y_train\'s shape : {y_train.shape}')
print(f'X_test\'s shape  : {X_test.shape}')
print(f'y_test\'s shape  : {y_test.shape}')

# Statistical info
print("\nStatistical info of training set:")
print(X_train.describe())
print("\nStatistical info of testing set:")
print(X_test.describe())
```

```
X_train's shape : (199364, 29)
y_train's shape : (199364,)
X_test's shape  : (85443, 29)
y_test's shape  : (85443,)
```




```
std          1.963554       1.657379       1.516716       1.417138
min        -46.855047     -61.344698     -33.680984      -5.568118
25%         -0.921539      -0.601213      -0.892838      -0.848835
50%          0.019705       0.063784       0.177888      -0.017852
75%          1.316707       0.802437       1.025520       0.745566
max          2.451888      22.057729       9.382558      16.715537

                 V5             V6             V7             V8  \
count  199364.000000  199364.000000  199364.000000  199364.000000
mean       -0.001494      -0.000218      -0.000879      -0.001080
std         1.368744       1.328673       1.226018       1.212338
min       -42.147898     -23.496714     -43.557242     -73.216718
25%        -0.692874      -0.769177      -0.554220      -0.209086
50%        -0.055832      -0.274307       0.039228       0.021803
75%         0.609349       0.397928       0.569638       0.327023
max        34.099309      23.917837      44.054461      20.007208

                 V9            V10  ...            V20            V21  \
count  199364.000000  199364.000000  ...  199364.000000  199364.000000
mean        0.000212       0.001357  ...       0.000430      -0.000014
std         1.102021       1.092801  ...       0.770257       0.743450
min       -13.434066     -24.588262  ...     -23.646890     -34.830382
25%        -0.644751      -0.535493  ...      -0.211662      -0.229272
50%        -0.049613      -0.092869  ...      -0.062880      -0.029065
75%         0.507096       0.458129  ...       0.133834       0.187095
max        15.594995      23.745136  ...      39.420904      27.202839

                 V22            V23            V24            V25  \
count  199364.000000  199364.000000  199364.000000  199364.000000
mean       -0.000022      -0.000258       0.000362       0.000395
std         0.727625       0.629145       0.605298       0.521175
min       -10.933144     -44.807735     -2.822684     -10.295397
25%        -0.544345      -0.162021      -0.354179      -0.316988
50%         0.006744      -0.010915       0.040974       0.010614
75%         0.531017       0.147583       0.438053       0.350802
max        10.503090      22.528412       4.022866       7.519589

                 V26            V27            V28     normAmount
count  199364.000000  199364.000000  199364.000000  199364.000000
mean       -0.000094      -0.000027       0.000015       0.001271
std         0.481842       0.401042       0.324840       0.981948
min        -2.534330     -22.565679     -11.710896      -0.353229
25%        -0.327127      -0.070864      -0.052907      -0.330640
50%        -0.052187       0.001064       0.011119      -0.265271
75%         0.241082       0.090491       0.077989      -0.043058
max         3.463246      12.152401      22.620072      78.235272

[8 rows x 29 columns]

Statistical info of testing set:
                 V1             V2             V3             V4             V5  \
count  85443.000000   85443.000000   85443.000000   85443.000000   85443.000000
mean      -0.000734       0.006277       0.003574      -0.001682       0.003486
std        1.947125       1.617050       1.515182       1.412908       1.406722
min      -56.407510     -72.715728     -48.325589      -5.683171    -113.743307
25%       -0.916858      -0.591858      -0.883828      -0.848202      -0.688288
50%        0.015238       0.070185       0.185047      -0.024180      -0.051627
75%        1.313257       0.806615       1.031155       0.737784       0.618067
max        2.454930      15.876923       4.079168      16.875344      34.801666
```



mean	-0.000734	0.000277	0.003574	-0.001682	0.003486
std	1.947325	1.637050	1.515182	1.412908	1.406722
min	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307
25%	-0.916858	-0.591858	-0.883828	-0.848282	-0.688280
50%	0.013238	0.070185	0.185047	-0.024109	-0.051627
75%	1.313257	0.806615	1.031155	0.737784	0.618867
max	2.454930	15.876923	4.079168	16.875344	34.801666

	V6	V7	V8	V9	V10 \
count	85443.000000	85443.000000	85443.000000	85443.000000	85443.000000
mean	0.000483	0.002030	0.004620	-0.000495	-0.001167
std	1.360636	1.262562	1.151291	1.090691	1.079574
min	-26.160506	-28.215112	-50.943369	-9.481456	-20.949192
25%	-0.766864	-0.553479	-0.207216	-0.638926	-0.535400
50%	-0.273686	0.042343	0.021782	-0.053821	-0.094949
75%	0.399864	0.572423	0.328337	0.597388	0.443126
max	73.301626	120.589494	18.748872	9.272376	15.331742

	...	V20	V21	V22	V23 \
count	...	85443.000000	85443.000000	85443.000000	85443.000000
mean	...	-0.001004	0.000833	0.000852	0.000602
std	...	0.772484	0.713266	0.721198	0.613394
min	...	-54.497720	-22.665685	-9.499423	-32.828995
25%	...	-0.211881	-0.226184	-0.537784	-0.161450
50%	...	-0.061529	-0.038687	0.000571	-0.011789
75%	...	0.133600	0.184846	0.523680	0.147023
max	...	38.117289	22.579714	7.228158	20.803344

	V24	V25	V26	V27	V28 \
count	85443.000000	85443.000000	85443.000000	85443.000000	85443.000000
mean	-0.000845	-0.000922	0.000220	0.000062	-0.000036
std	0.606464	0.521520	0.483126	0.409616	0.341987
min	-2.836627	-8.696627	-2.604551	-9.793568	-15.430084
25%	-0.355671	-0.318716	-0.326068	-0.070797	-0.053129
50%	0.040976	0.013508	-0.051695	0.001984	0.011561
75%	0.441093	0.350817	0.240657	0.092224	0.078000
max	4.584549	5.826159	3.517346	31.612198	33.847808

	normAmount
count	85443.000000
mean	-0.002966
std	1.036492
min	-0.353229
25%	-0.331288
50%	-0.265171
75%	-0.047356
max	102.362243

[8 rows x 29 columns]

jupyter 10.2HD sample Last Checkpoint: 1 hour ago

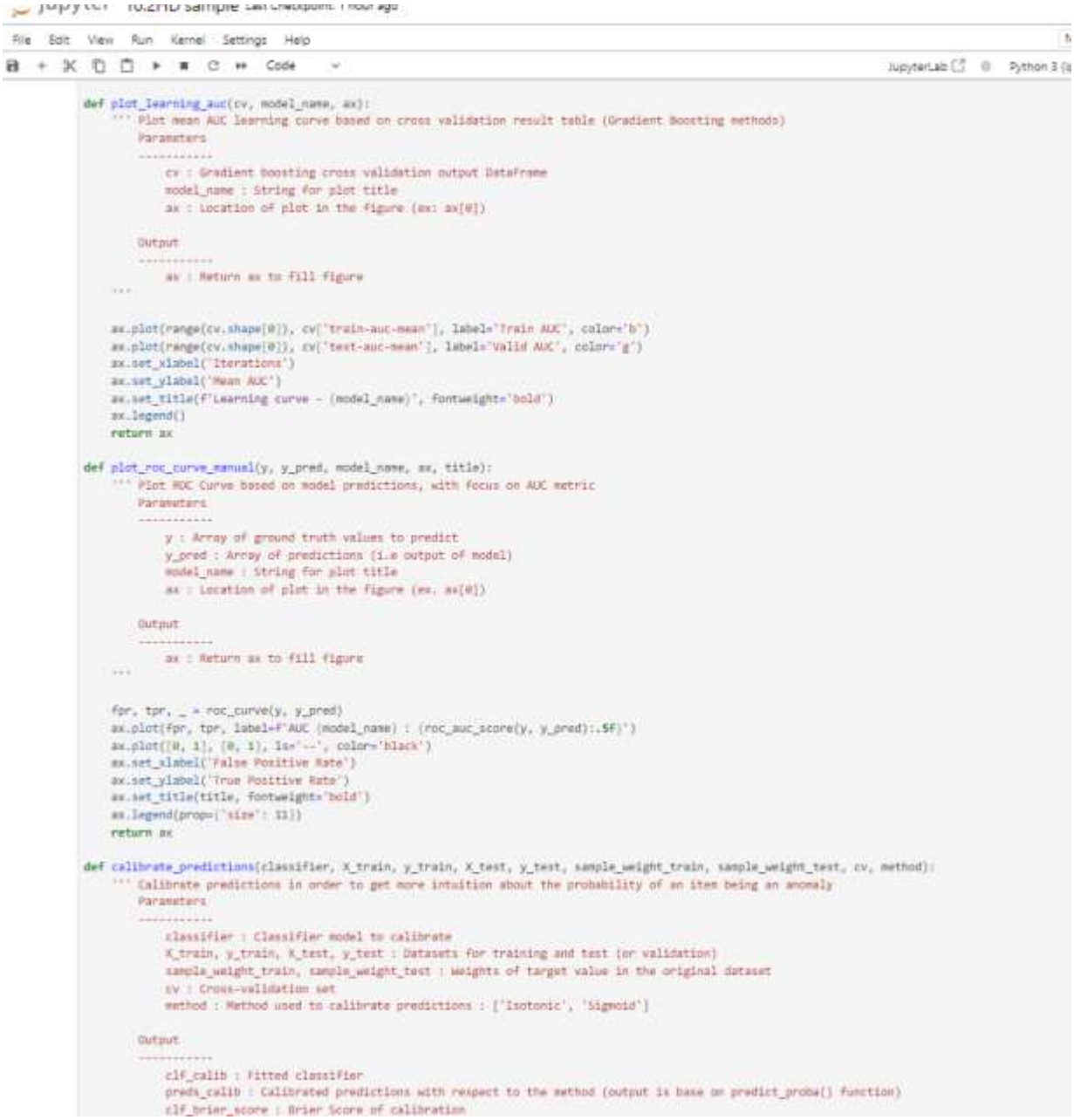
File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (pykernel)

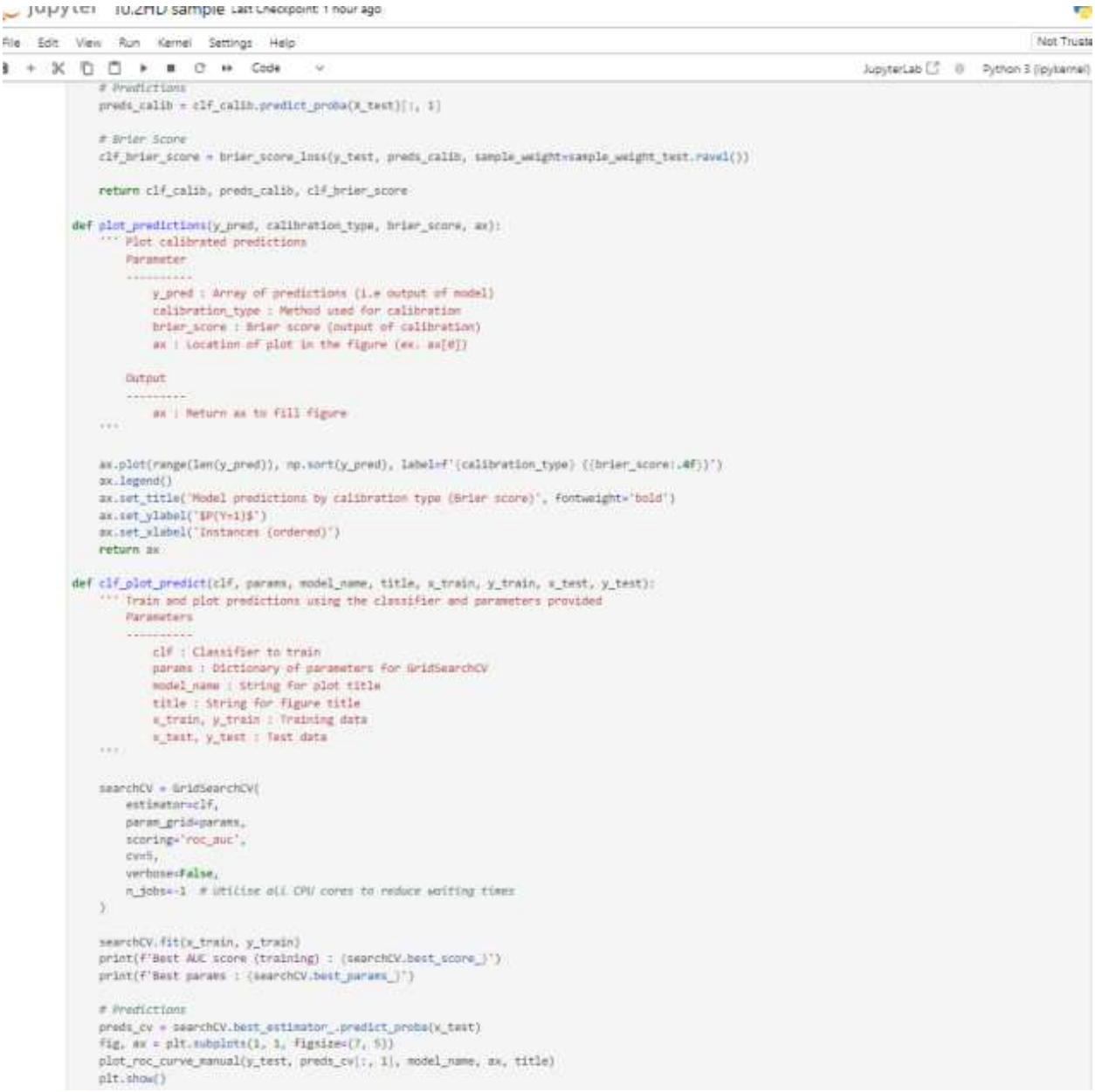
1.4 Functions

Here we define a few functions that we will use later in the project.

[7]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_curve, roc_auc_score, brier_score_loss
from sklearn.model_selection import GridSearchCV

def apply_PCA(X, plot=True):
    ''' Fit PCA on data, then plot explained variance if needed
        Parameter
        ----------
            X : Dataframe that will be scaled and transformed according to PCA
            plot : Boolean. If True, displays the explained variance by principal components of PCA

        Output
        -------
            Array of principal components resulting from PCA
            Vector of explained variance
    '''

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # By default keeps all components
    pca = PCA()
    pca_comp = pca.fit_transform(X_scaled)
    explained_var = pca.explained_variance_ratio_

    if plot:
        fig, ax = plt.subplots(1, 1, figsize=(10, 5))
        ax.plot(range(0, len(explained_var)), explained_var.cumsum(), marker='o')
        ax.set(xlabel='Principal components', ylabel='Inertia')
        plt.axvline(x=11, color='r', linestyle='--')
        plt.title('Amount of inertia explained by principal components', fontweight='bold')
        plt.show()

    return pca_comp, explained_var

def plot_pca_3D(pca_components, labels):
    ''' Plot values of principal components of PCA
        Parameters
        -----------
            pca_components : Array resulting from fitting PCA to data
            labels : ground truth labels from data
    '''

    not_fraud = pca_components[labels['Class'] == 0]
    fraud = pca_components[labels['Class'] == 1]

    plot3D = plt.figure().add_subplot(projection='3d')

    # First, second and third principal components in each x, y and z variable
    plot3D.scatter(fraud[:, 2], fraud[:, 0], fraud[:, 1], label='Fraudulent transactions', color='r', alpha=1)
    plot3D.scatter(not_fraud[:, 2], not_fraud[:, 0], not_fraud[:, 1], label='Not fraudulent', color='g', alpha=0.2)
```



```
def plot_learning_auc(cv, model_name, ax):
    ''' Plot mean AUC learning curve based on cross validation result table (Gradient Boosting methods)
        Parameters
        ----------
            cv : Gradient boosting cross validation output DataFrame
            model_name : String for plot title
            ax : Location of plot in the figure (ex: ax[0])

        Output
        ----------
            ax : Return ax to fill figure
    '''

    ax.plot(range(cv.shape[0]), cv['train-auc-mean'], label='Train AUC', color='b')
    ax.plot(range(cv.shape[0]), cv['test-auc-mean'], label='Valid AUC', color='g')
    ax.set_xlabel('Iterations')
    ax.set_ylabel('Mean AUC')
    ax.set_title(f'Learning curve - {model_name}', fontweight='bold')
    ax.legend()
    return ax

def plot_roc_curve_manual(y, y_pred, model_name, ax, title):
    ''' Plot ROC Curve based on model predictions, with focus on AUC metric
        Parameters
        ----------
            y : Array of ground truth values to predict
            y_pred : Array of predictions (i.e output of model)
            model_name : String for plot title
            ax : Location of plot in the figure (ex. ax[0])

        Output
        ----------
            ax : Return ax to fill figure
    '''

    fpr, tpr, _ = roc_curve(y, y_pred)
    ax.plot(fpr, tpr, label=f'AUC {model_name} : {roc_auc_score(y, y_pred):.5f}')
    ax.plot([0, 1], [0, 1], ls='--', color='black')
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title(title, fontweight='bold')
    ax.legend(prop={'size': 11})
    return ax

def calibrate_predictions(classifier, X_train, y_train, X_test, y_test, sample_weight_train, sample_weight_test, cv, method):
    ''' Calibrate predictions in order to get more intuition about the probability of an item being an anomaly
        Parameters
        ----------
            classifier : Classifier model to calibrate
            X_train, y_train, X_test, y_test : Datasets for training and test (or validation)
            sample_weight_train, sample_weight_test : Weights of target value in the original dataset
            cv : Cross-validation set
            method : Method used to calibrate predictions : ['Isotonic', 'Sigmoid']

        Output
        ----------
            clf_calib : Fitted classifier
            preds_calib : Calibrated predictions with respect to the method (output is base on predict_proba() function)
            clf_brier_score : Brier Score of calibration
```



```
    # Predictions
    preds_calib = clf_calib.predict_proba(X_test)[:, 1]

    # Brier Score
    clf_brier_score = brier_score_loss(y_test, preds_calib, sample_weight=sample_weight_test.ravel())

    return clf_calib, preds_calib, clf_brier_score

def plot_predictions(y_pred, calibration_type, brier_score, ax):
    ''' Plot calibrated predictions
        Parameter
        ----------
            y_pred : Array of predictions (i.e output of model)
            calibration_type : Method used for calibration
            brier_score : Brier score (output of calibration)
            ax : Location of plot in the figure (ex. ax[0])

        Output
        ---------
            ax : Return ax to fill figure
    '''

    ax.plot(range(len(y_pred)), np.sort(y_pred), label=f'{calibration_type} ({brier_score:.4f})')
    ax.legend()
    ax.set_title('Model predictions by calibration type (Brier score)', fontweight='bold')
    ax.set_ylabel('$P(Y=1)$')
    ax.set_xlabel('Instances (ordered)')
    return ax

def clf_plot_predict(clf, params, model_name, title, x_train, y_train, x_test, y_test):
    ''' Train and plot predictions using the classifier and parameters provided
        Parameters
        ----------
            clf : Classifier to train
            params : Dictionary of parameters for GridSearchCV
            model_name : String for plot title
            title : String for figure title
            x_train, y_train : Training data
            x_test, y_test : Test data
    '''

    searchCV = GridSearchCV(
        estimator=clf,
        param_grid=params,
        scoring='roc_auc',
        cv=5,
        verbose=False,
        n_jobs=-1  # Utilise all CPU cores to reduce waiting times
    )

    searchCV.fit(x_train, y_train)
    print(f'Best AUC score (training) : {searchCV.best_score_}')
    print(f'Best params : {searchCV.best_params_}')

    # Predictions
    preds_cv = searchCV.best_estimator_.predict_proba(x_test)
    fig, ax = plt.subplots(1, 1, figsize=(7, 5))
    plot_roc_curve_manual(y_test, preds_cv[:, 1], model_name, ax, title)
    plt.show()
```

[9]:

```
import pandas as pd

# Assuming the datasets are already loaded as X_train, X_test, y_train, y_test
# Example:
# X_train = pd.read_csv('X_train.csv')
# X_test = pd.read_csv('X_test.csv')
# y_train = pd.read_csv('y_train.csv')
# y_test = pd.read_csv('y_test.csv')

# Checking for missing values in the datasets
print(f'Missing values in X_train : {X_train.isnull().sum().sum()}')
print(f'Missing values in y_train : {y_train.isnull().sum().sum()}')
print(f'Missing values in X_test  : {X_test.isnull().sum().sum()}')
print(f'Missing values in y_test  : {y_test.isnull().sum().sum()}')
```

```
Missing values in X_train : 0
Missing values in y_train : 0
Missing values in X_test  : 0
Missing values in y_test  : 0
```

Code JupyterLab Python 3 (ipykern

Duplicated values

Some values are duplicated in the dataset, and we will focus on them in the *Feature Engineering* part.

[11]:
```
import pandas as pd

# Assuming the datasets are already loaded as X_train, X_test, y_train, y_test
# Example:
# X_train = pd.read_csv('X_train.csv')
# X_test = pd.read_csv('X_test.csv')
# y_train = pd.read_csv('y_train.csv')
# y_test = pd.read_csv('y_test.csv')

# Checking for Duplicate Values in the datasets
print(f'Duplicated values in X_train : {X_train.duplicated().sum()}')
print(f'Duplicated values in y_train : {y_train.duplicated().sum()}')
print(f'Duplicated values in X_test  : {X_test.duplicated().sum()}')
print(f'Duplicated values in y_test  : {y_test.duplicated().sum()}')
```

```
Duplicated values in X_train : 5161
Duplicated values in y_train : 199362
Duplicated values in X_test  : 1375
Duplicated values in y_test  : 85441
```

Imbalanced classes

We can see that the classes are severely imbalanced in the overall dataset, with fraudulent transactions accounting for only 492 samples in comparison to regular transactions at 284315 samples. This means fraudulent transactions only account for 0.173% out of the total transactions.

[13]:
```
import pandas as pd

# Assuming y_train and y_test contain the target labels, where 1 represents the anomaly/fraud and 0 represents normal transactions

# Checking class distribution in y_train
class_distribution_train = y_train.value_counts(normalize=True)
class_distribution_test = y_test.value_counts(normalize=True)

# Printing the imbalance in percentages
print(f'Class distribution in y_train:\n{class_distribution_train * 100}\n')
print(f'Class distribution in y_test:\n{class_distribution_test * 100}\n')

# Checking total counts
fraud_train = y_train.value_counts()[1]
not_fraud_train = y_train.value_counts()[0]
fraud_test = y_test.value_counts()[1]
not_fraud_test = y_test.value_counts()[0]

print(f'Total samples in y_train: {len(y_train)}')
print(f'Fraudulent transactions in y_train: {fraud_train} ({(fraud_train / len(y_train)) * 100:.2f}%)')
print(f'Regular transactions in y_train: {not_fraud_train} ({(not_fraud_train / len(y_train)) * 100:.2f}%)\n')

print(f'Total samples in y_test: {len(y_test)}')
print(f'Fraudulent transactions in y_test: {fraud_test} ({(fraud_test / len(y_test)) * 100:.2f}%)')
print(f'Regular transactions in y_test: {not_fraud_test} ({(not_fraud_test / len(y_test)) * 100:.2f}%)\n')
```

```
Class distribution in y_train:
0    99.82695
1     0.17305
Name: proportion, dtype: float64

Class distribution in y_test:
0    99.827955
1     0.172045
Name: proportion, dtype: float64

Total samples in y_train: 199364
Fraudulent transactions in y_train: 345 (0.17%)
Regular transactions in y_train: 199019 (99.83%)

Total samples in y_test: 85443
Fraudulent transactions in y_test: 147 (0.17%)
Regular transactions in y_test: 85296 (99.83%)
```

The training and test sets both contain 29 features, and these features are the same.

[15]:
```
# Check if features in the training and test sets are the same
train_features = X_train.columns
test_features = X_test.columns

if train_features.equals(test_features):
    print("The training and test sets contain the same features.")
    print(f"Number of features in train and test sets: {len(train_features)}")
else:
    print("The training and test sets have different features.")
    print(f"Features in training set: {list(train_features)}")
    print(f"Features in test set: {list(test_features)}")
```

```
The training and test sets contain the same features.
Number of features in train and test sets: 29
```

jupyter 10.2MB sample Last Checkpoint: 1 hour ago

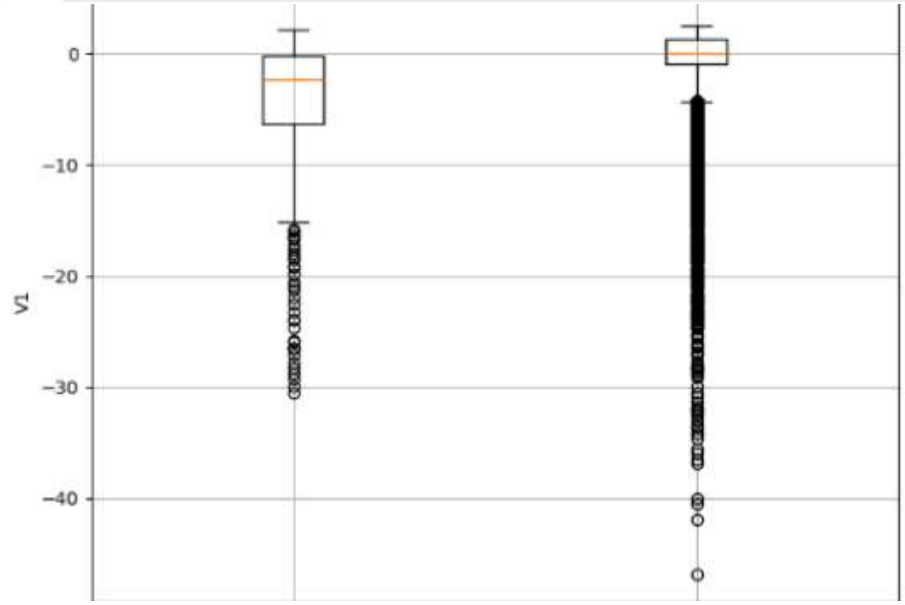
File Edit View Run Kernel Settings Help

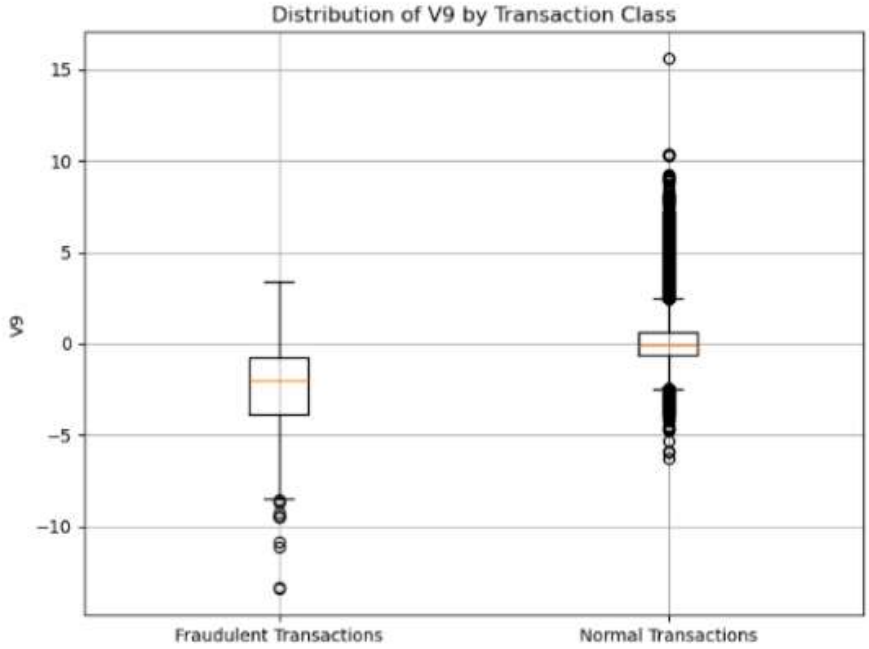
Code JupyterLab Python 3 (ipyk

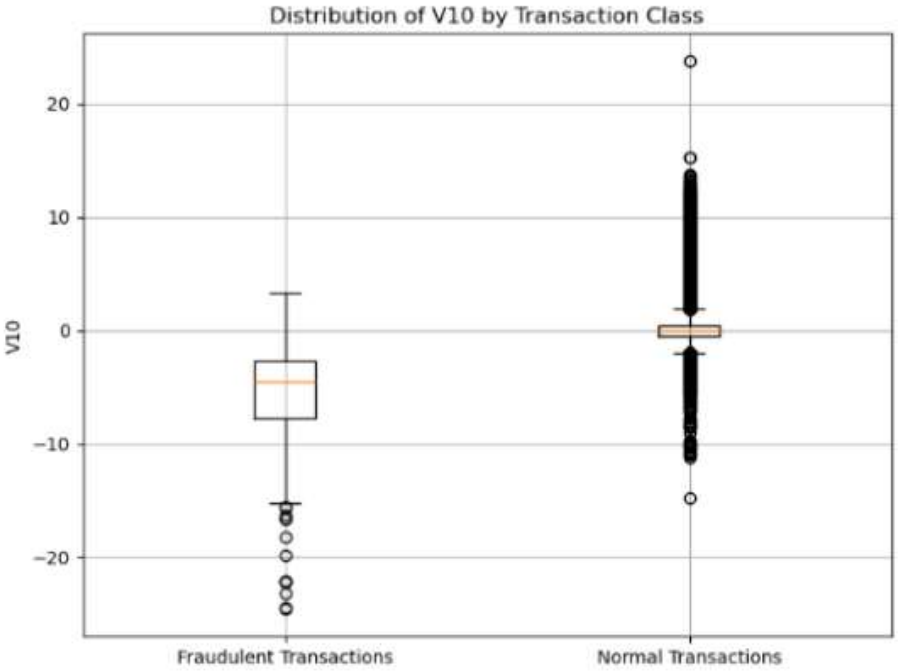
Features distributions by classes

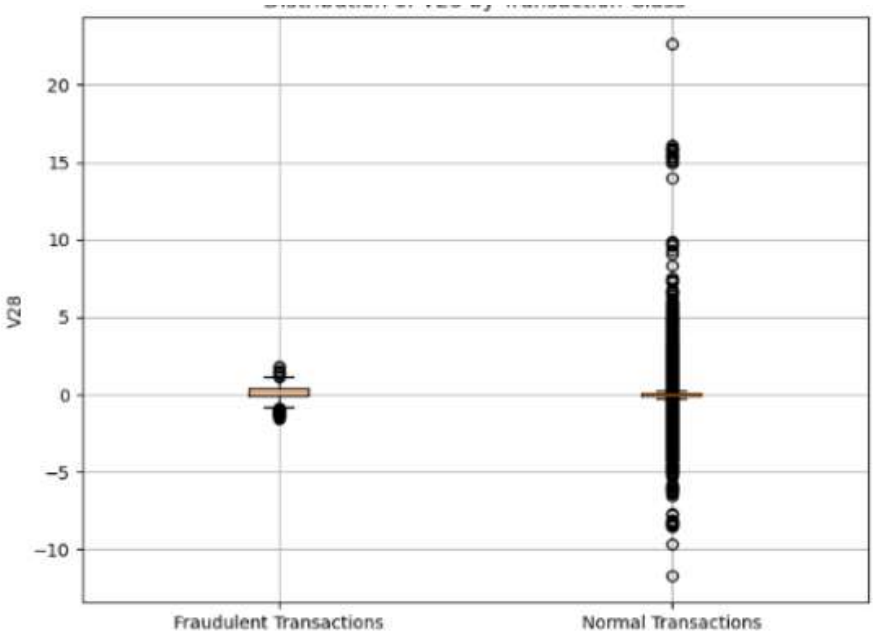
We take a quick look a the features distribution by target class. By comparing boxplots, we would be able to spot features which have specific distribution if the item is fraudulent or not. In n general, we can see that many of the normal samples have significant outlier values.

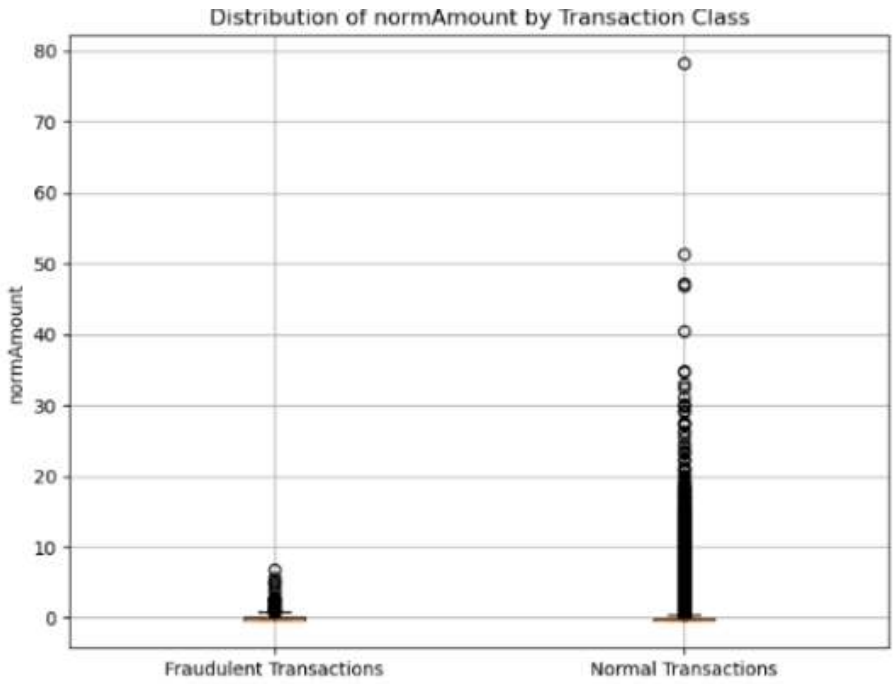
[17]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming X_train and y_train are already defined and contain your training data
# y_train should be a Series with 1 for fraudulent and 0 for normal transactions

# Ensure y_train is treated as a Series and get its values
if isinstance(y_train, pd.Series):
    y_train_values = y_train.values
else:
    y_train_values = y_train['Class'].values  # Assuming y_train is a DataFrame

# Separate fraudulent and benign transactions
fraudulent = X_train[y_train_values == 1]
benign = X_train[y_train_values == 0]

# Loop through all features to create boxplots
for feature in X_train.columns:
    # Check if the feature is numeric
    if pd.api.types.is_numeric_dtype(X_train[feature]):
        fig, ax = plt.subplots(1, 1, figsize=(8, 6))
        plt.boxplot([fraudulent[feature], benign[feature]])
        plt.title(f'Distribution of {feature} by Transaction Class')
        plt.xticks([1, 2], ['Fraudulent Transactions', 'Normal Transactions'])
        plt.ylabel(feature)  # Add y-axis label for clarity
        plt.grid()  # Optional: Add grid for better readability
        plt.show()
```




Distribution of V9 by Transaction Class
V9
15
10
5
0
−5
−10
Fraudulent Transactions
Normal Transactions


Distribution of V10 by Transaction Class
V10
20
10
0
−10
−20
Fraudulent Transactions
Normal Transactions


V28
20
15
10
5
0
−5
−10
Fraudulent Transactions
Normal Transactions


Distribution of normAmount by Transaction Class
normAmount
80
70
60
50
40
30
20
10
0
Fraudulent Transactions
Normal Transactions

jupyter 10.2HD sample Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help

Code

JupyterLab Python 3 (ipyk

2.2 Feature correlations

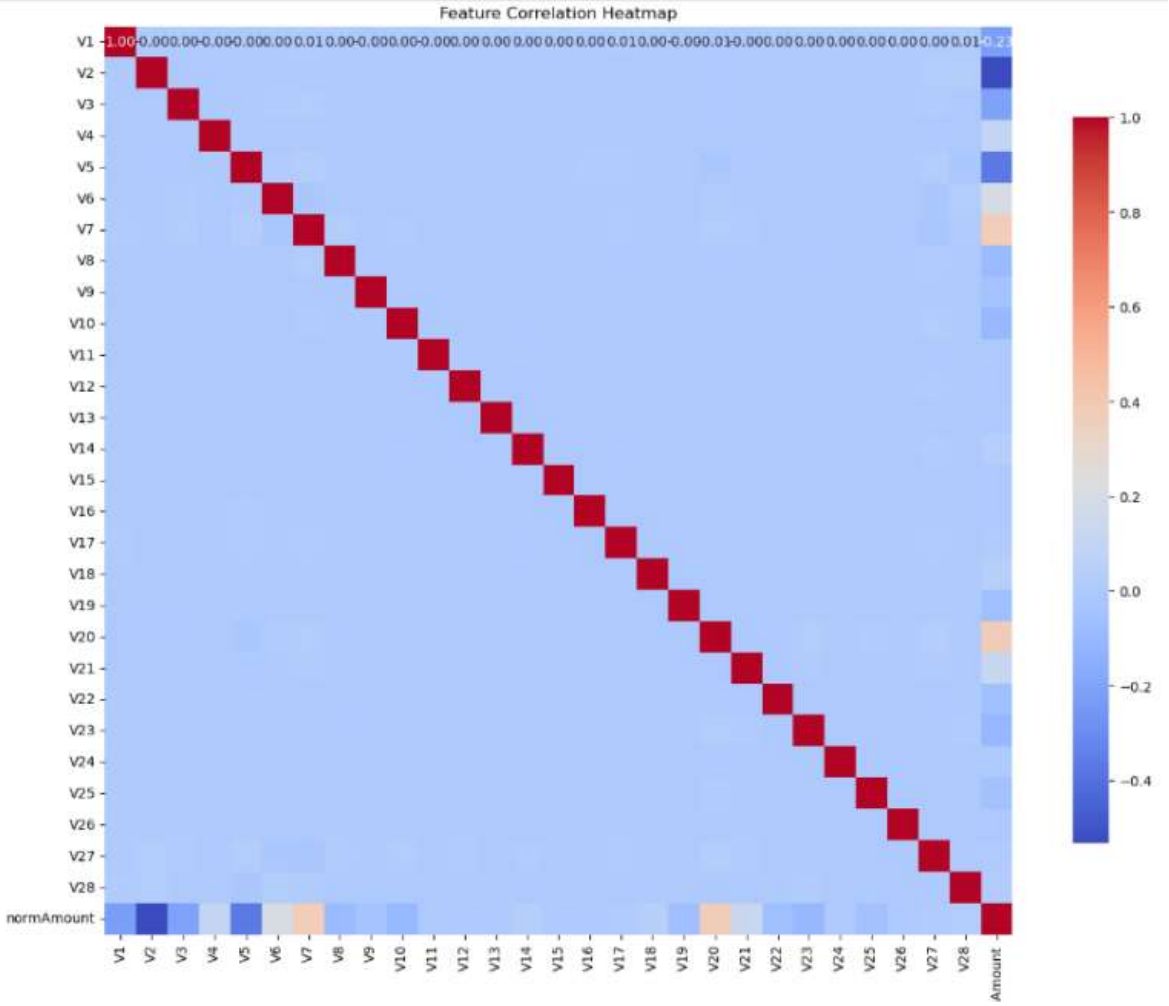
It can be interesting to get an intuition about the correlations between features, and the correlations between each feature and the target. We can see that the most divers correlations seem to be related to normAccount and every other feature.

[19]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming X_train is already defined and contains your training data

# Calculate the correlation matrix
corr = X_train.corr()

# Create a heatmap to visualize the correlations
plt.figure(figsize=(16, 12))
sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm', square=True, cbar_kws={"shrink": .8})
plt.title('Feature Correlation Heatmap')
plt.show()
```

Code

JupyterLab Python 3 (ipykernel)

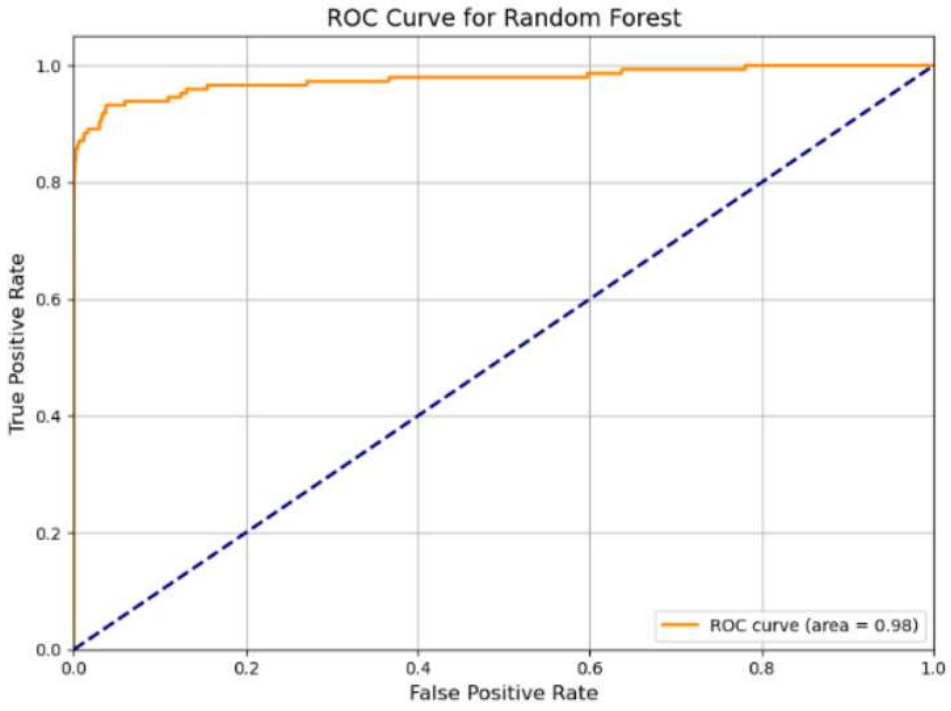


Feature correlation with target

Roughly half of the features have a somewhat significant correlation to the targets, but none are highly correlated.

[21]:
```
# Ensure y_train is a DataFrame with the correct column name
if isinstance(y_train, pd.Series):
    y_train = y_train.to_frame(name='Class')  # Convert Series to DataFrame with 'Class' column

# Concatenate X_train and y_train
df_combined = pd.concat([X_train, y_train], axis=1)

# Check if 'Class' column exists in the concatenated DataFrame
if 'Class' not in df_combined.columns:
    print("Error: 'Class' column not found in concatenated DataFrame")

# Calculate absolute correlations with the target variable 'Class'
corr_target = np.abs(df_combined.corr()['Class']).sort_values(ascending=False)[1:]

# Plotting the bar plot for feature correlations
fig, ax = plt.subplots(1, 1, figsize=(12, 5))
corr_target.plot(kind='bar', ax=ax)
plt.title('Features Correlation with Target', fontweight='bold')
plt.ylabel('Correlation with Target')
plt.xlabel('Feature')
plt.show()
```

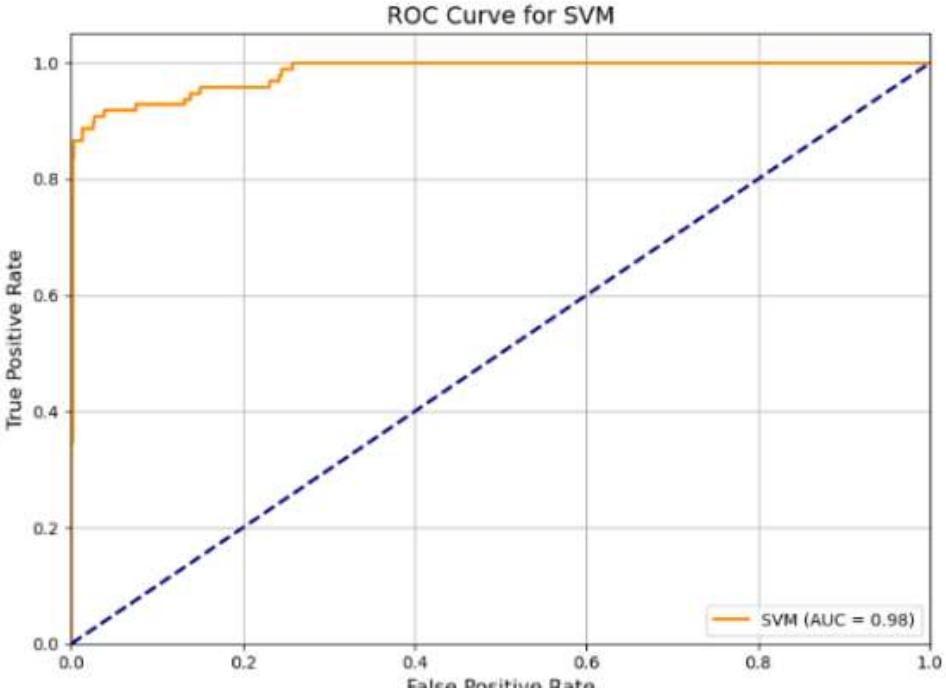
```python
# Importing necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, roc_curve, auc
import matplotlib.pyplot as plt

# Manually specifying random forest model parameters
rf_model = RandomForestClassifier(n_estimators=50, max_depth=5, class_weight='balanced', random_state=0)

# Fitting the model to training data
rf_model.fit(X_train, y_train)

# Get predicted probabilities for test set
y_test_pred_proba_rf = rf_model.predict_proba(X_test)[:, 1]

# Calculate ROC-AUC score
roc_auc_rf = roc_auc_score(y_test, y_test_pred_proba_rf)
print("ROC AUC score for Random Forest:", roc_auc_rf)

# Plot ROC curve
fpr, tpr, _ = roc_curve(y_test, y_test_pred_proba_rf)
roc_auc = auc(fpr, tpr)

# Plot ROC Curve
plt.figure(figsize=(8, 6))  # Increase figure size for better visibility
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  # Diagonal line for random chance
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title('ROC Curve for Random Forest', fontsize=14)
plt.legend(loc="lower right")
plt.grid(True)  # Adding a grid to make the plot clearer
plt.tight_layout()  # Adjust layout for better visibility of all elements
plt.show()
```



```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_auc_score, roc_curve, auc
import time

# Example dataset (replace with your actual dataset)
# X, y = pd.read_csv('your_dataset.csv'), pd.read_csv('your_labels.csv')

# Splitting dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define hyperparameter grid for SVM (using SGDClassifier)
params = {
    'loss': ['modified_huber'],
    'alpha': [0.00001, 0.0001, 0.001, 0.01, 0.1],
    'epsilon': [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1],
    'learning_rate': ['invscaling'],
    'eta0': [0.00001, 0.0001, 0.001],
    'class_weight': ['balanced']
}

# Initialize GridSearchCV
grid_search = GridSearchCV(SGDClassifier(random_state=0, fit_intercept=False),
                           param_grid=params, cv=3, scoring='roc_auc', n_jobs=-1, verbose=1)

# Start timer
start_time = time.time()

# Fit GridSearchCV
print("Starting grid search...")
grid_search.fit(X_train, y_train)
print("Grid search completed.")

# Output best parameters from grid search
best_params = grid_search.best_params_
print("Grid search best params")
print(best_params)

# Using best params to build SVM model
best_svm = SGDClassifier(**best_params, random_state=0, fit_intercept=False)
best_svm.fit(X_train, y_train)

# Calculate AUC score on training data
y_train_pred_proba = best_svm.decision_function(X_train)
best_auc_score = roc_auc_score(y_train, y_train_pred_proba)

# Display best AUC score and parameters
print("Using best grid search params to build SVM model.")
print(f"Best AUC score (training) : {best_auc_score}")
print(f"Best params : {best_params}")
```

```
print(f"CPU times: total: {cpu_time:.1f} s")

# Plotting ROC Curve
y_test_pred_proba = best_svm.decision_function(X_test)
fpr, tpr, _ = roc_curve(y_test, y_test_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))  # Increase figure size
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'SVM (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  # Diagonal line for random chance
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title('ROC Curve for SVM', fontsize=14)
plt.legend(loc="lower right")
plt.grid(True)  # Adding a grid to make the plot clearer
plt.tight_layout()  # Adjust layout for better visibility of all elements
plt.show()
```

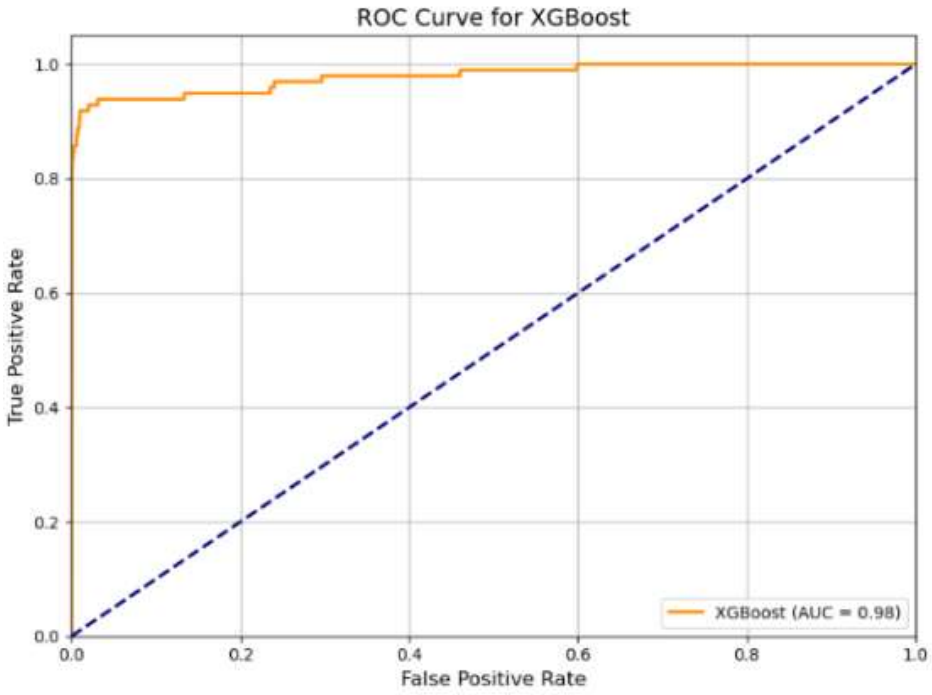
```
Starting grid search...
Fitting 3 folds for each of 105 candidates, totalling 315 fits
Grid search completed.
Grid search best params
{'alpha': 0.1, 'class_weight': 'balanced', 'epsilon': 1e-06, 'eta0': 1e-05, 'learning_rate': 'invscaling', 'loss': 'modified_huber'}
Using best grid search params to build SVM model.
Best AUC score (training) : 0.979084578645343
Best params : {'alpha': 0.1, 'class_weight': 'balanced', 'epsilon': 1e-06, 'eta0': 1e-05, 'learning_rate': 'invscaling', 'loss': 'modified_hube
CPU times: total: 43.9 s
```



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV, train_test_split
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score, roc_curve, auc
import time

# Example dataset (replace with your actual dataset)
# X, y = pd.read_csv('your_dataset.csv'), pd.read_csv('your_labels.csv')

# Splitting dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define hyperparameter grid for XGBoost
params = {
    'max_depth': [4],
    'learning_rate': [0.00001, 0.0001, 0.001],
    'gamma': [0.1],
    'subsample': [0.2, 0.3, 0.4, 0.5],
    'objective': ['binary:logistic'],
    'eval_metric': ['auc'],
    'scale_pos_weight': [577],  # Value is sum(negative class)/sum(positive class)
    'n_estimators': [80]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(XGBClassifier(booster='gbtree'), param_grid=params, cv=5, scoring='roc_auc', n_jobs=-1, verbose=1)

# Start timer
start_time = time.time()

# Fit GridSearchCV
print("Starting grid search...")
grid_search.fit(X_train, y_train)
print("Grid search completed.")

# Output best parameters from grid search
best_params = grid_search.best_params_
print("Grid search best params")
print(best_params)

# Using best params to build XGBoost model
xgb_model = XGBClassifier(
    max_depth=best_params['max_depth'],
    learning_rate=best_params['learning_rate'],  # Use the value directly
    gamma=best_params['gamma'],  # Use the value directly
    subsample=best_params['subsample'],  # Use the value directly
    objective='binary:logistic',
    eval_metric='auc',
    scale_pos_weight=577,
    n_estimators=80
)

# Fit the XGBoost model
```

```
print("Best AUC score (training) for XGBoost:", roc_auc_xgb_train)

# Plotting ROC Curve
y_test_pred_proba = xgb_model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_test_pred_proba)
roc_auc_xgb = auc(fpr, tpr)

plt.figure(figsize=(8, 6))  # Increase figure size for better visibility
plt.plot(fpr, tpr, color='darkorange', lw=2, label='XGBoost (AUC = %0.2f)' % roc_auc_xgb)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  # Diagonal line for random chance
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title('ROC Curve for XGBoost', fontsize=14)
plt.legend(loc="lower right")
plt.grid(True)  # Adding a grid for clarity
plt.tight_layout()  # Adjust layout for better visibility of all elements
plt.show()
```

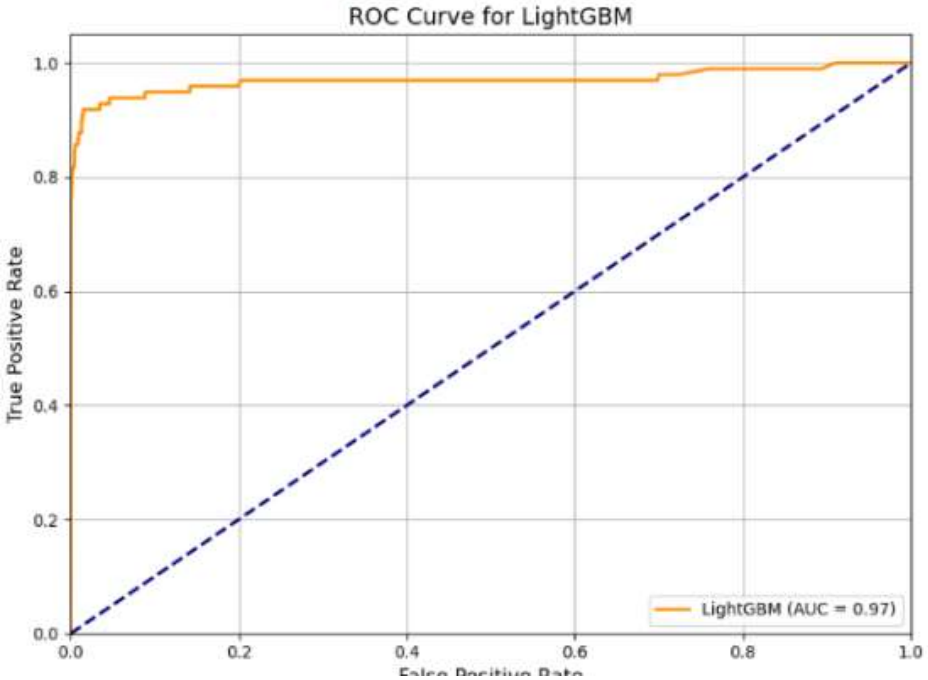
```
Starting grid search...
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Grid search completed.
Grid search best params
{'eval_metric': 'auc', 'gamma': 0.1, 'learning_rate': 0.0001, 'max_depth': 4, 'n_estimators': 80, 'objective': 'binary:logistic', 'scale_pos_weigh
7, 'subsample': 0.4}
Best AUC score (training) for XGBoost: 0.9952162620087504
```



```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV, train_test_split
from lightgbm import LGBMClassifier
from sklearn.metrics import roc_auc_score, roc_curve, auc
import time

# Example dataset (replace with your actual dataset)
# X, y = pd.read_csv('your_dataset.csv'), pd.read_csv('your_labels.csv')

# Splitting dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define hyperparameter grid for LightGBM
params = {
    'learning_rate': [0.0001],
    'max_depth': [9],
    'reg_alpha': [0.001],
    'reg_lambda': [0.001],
    'subsample': [0.0001],
    'objective': ['binary'],
    'scale_pos_weight': [577],
    'n_estimators': [60]
}

# Initialize GridSearchCV for LightGBM
grid_search = GridSearchCV(LGBMClassifier(random_state=0), param_grid=params, cv=5, scoring='roc_auc', n_jobs=-1, verbose=1)

# Start timer
start_time = time.time()

# Fit GridSearchCV
print("Starting grid search...")
grid_search.fit(X_train, y_train)
print("Grid search completed.")

# Output best parameters from grid search
best_params = grid_search.best_params_
print("Grid search best params")
print(best_params)

# Using best params to build LightGBM model
lgbm_model = LGBMClassifier(
    learning_rate=best_params['learning_rate'],  # Use the value directly
    max_depth=best_params['max_depth'],  # Use the value directly
    reg_alpha=best_params['reg_alpha'],  # Use the value directly
    reg_lambda=best_params['reg_lambda'],  # Use the value directly
    subsample=best_params['subsample'],  # Use the value directly
    objective='binary',
    scale_pos_weight=577,
    n_estimators=best_params['n_estimators']  # Use the value directly
)

# Fit the LightGBM model
```

```
plt.legend(loc="lower right")
plt.grid(True)  # Adding a grid for clarity
plt.tight_layout()  # Adjust layout for better visibility of all elements
plt.show()
```

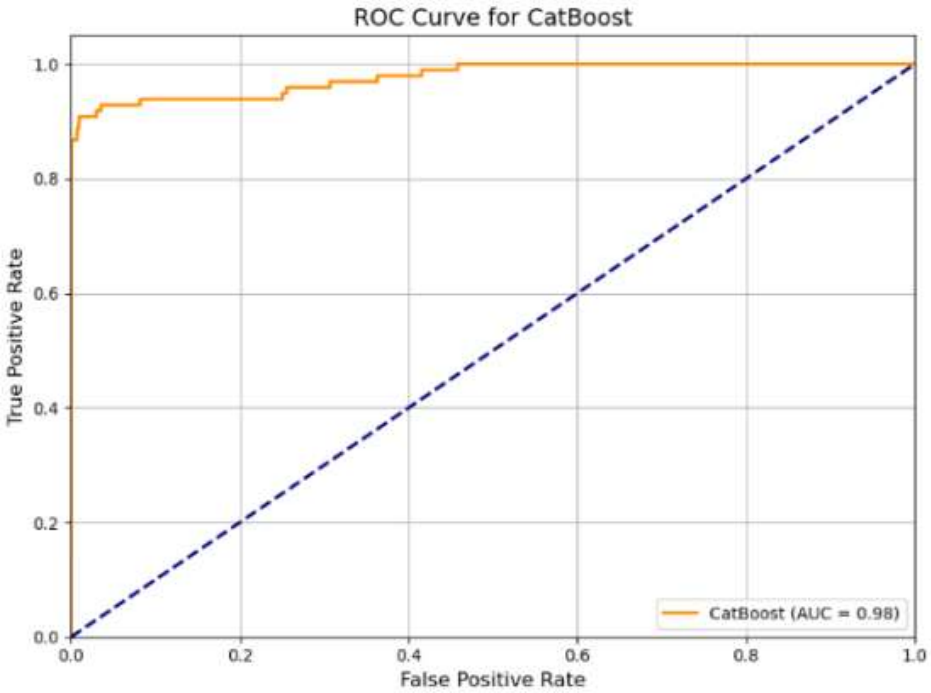
```
Starting grid search...
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[LightGBM] [Info] Number of positive: 394, number of negative: 227451
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.015376 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7395
[LightGBM] [Info] Number of data points in the train set: 227845, number of used features: 29
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.001729 -> initscore=-6.358339
[LightGBM] [Info] Start training from score -6.358339
Grid search completed.
Grid search best params
{'learning_rate': 0.0001, 'max_depth': 9, 'n_estimators': 60, 'objective': 'binary', 'reg_alpha': 0.001, 'reg_lambda': 0.001, 'scale_pos_weight':
'subsample': 0.0001}
[LightGBM] [Info] Number of positive: 394, number of negative: 227451
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.016902 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7395
[LightGBM] [Info] Number of data points in the train set: 227845, number of used features: 29
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.001729 -> initscore=-6.358339
[LightGBM] [Info] Start training from score -6.358339
Best AUC score (training) for LightGBM: 0.9996851138096415
```



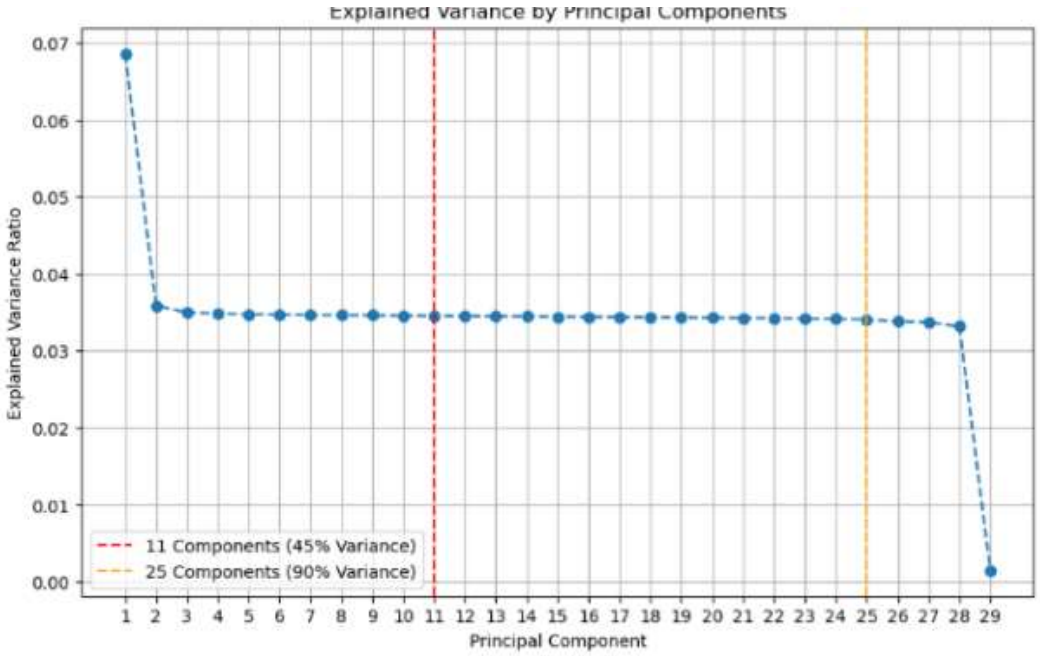
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV, train_test_split
from catboost import CatBoostClassifier
from sklearn.metrics import roc_auc_score, roc_curve, auc
import time

# Example dataset (replace with your actual dataset)
# X, y = pd.read_csv('your_dataset.csv'), pd.read_csv('your_labels.csv')

# Splitting dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define hyperparameter grid for CatBoost
params = {
    'learning_rate': [0.001],
    'max_depth': [3],
    'reg_lambda': [0.001],
    'bootstrap_type': ['Bernoulli'],
    'subsample': [0.1],
    'eval_metric': ['AUC'],
    'scale_pos_weight': [577],
    'logging_level': ['Silent'],
    'n_estimators': [90]
}

# Initialize GridSearchCV for CatBoost
grid_search = GridSearchCV(CatBoostClassifier(random_state=0), param_grid=params, cv=5, scoring='roc_auc', n_jobs=-1, verbose=1)

# Start timer
start_time = time.time()

# Fit GridSearchCV
print("Starting grid search...")
grid_search.fit(X_train, y_train)
print("Grid search completed.")

# Output best parameters from grid search
best_params = grid_search.best_params_
print("Grid search best params")
print(best_params)

# Best score from grid search
print("Best AUC score from grid search:", grid_search.best_score_)

# Using best params to build CatBoost model
cat_model = CatBoostClassifier(
    learning_rate=best_params['learning_rate'],  # Use the value directly
    max_depth=best_params['max_depth'],  # Use the value directly
    reg_lambda=best_params['reg_lambda'],  # Use the value directly
    bootstrap_type=best_params['bootstrap_type'],  # Use the value directly
    subsample=best_params['subsample'],  # Use the value directly
    #eval_metric='AUC',  # Directly assign since it's a string
    scale_pos_weight=577,
```

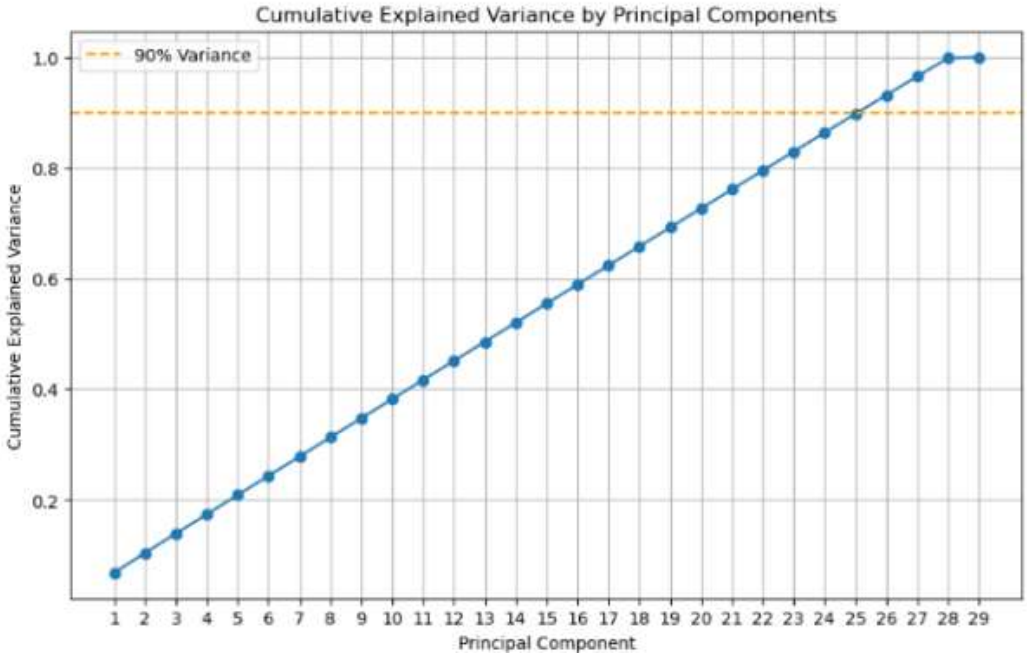
```
# Plotting ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_test_pred_proba)
roc_auc_cat_test = auc(fpr, tpr)

plt.figure(figsize=(8, 6))  # Increase figure size for better visibility
plt.plot(fpr, tpr, color='darkorange', lw=2, label='CatBoost (AUC = %0.2f)' % roc_auc_cat_test)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  # Diagonal line for random chance
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title('ROC Curve for CatBoost', fontsize=14)
plt.legend(loc="lower right")
plt.grid(True)  # Adding a grid for clarity
plt.tight_layout()  # Adjust layout for better visibility of all elements
plt.show()
```

```
Starting grid search...
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Grid search completed.
Grid search best params
{'bootstrap_type': 'Bernoulli', 'eval_metric': 'AUC', 'learning_rate': 0.001, 'logging_level': 'Silent', 'max_depth': 3, 'n_estimators': 90, 'reg_
a': 0.001, 'scale_pos_weight': 577, 'subsample': 0.1}
Best AUC score from grid search: 0.9784795899452634
Best AUC score (testing) for CatBoost: 0.9771116979431052
```

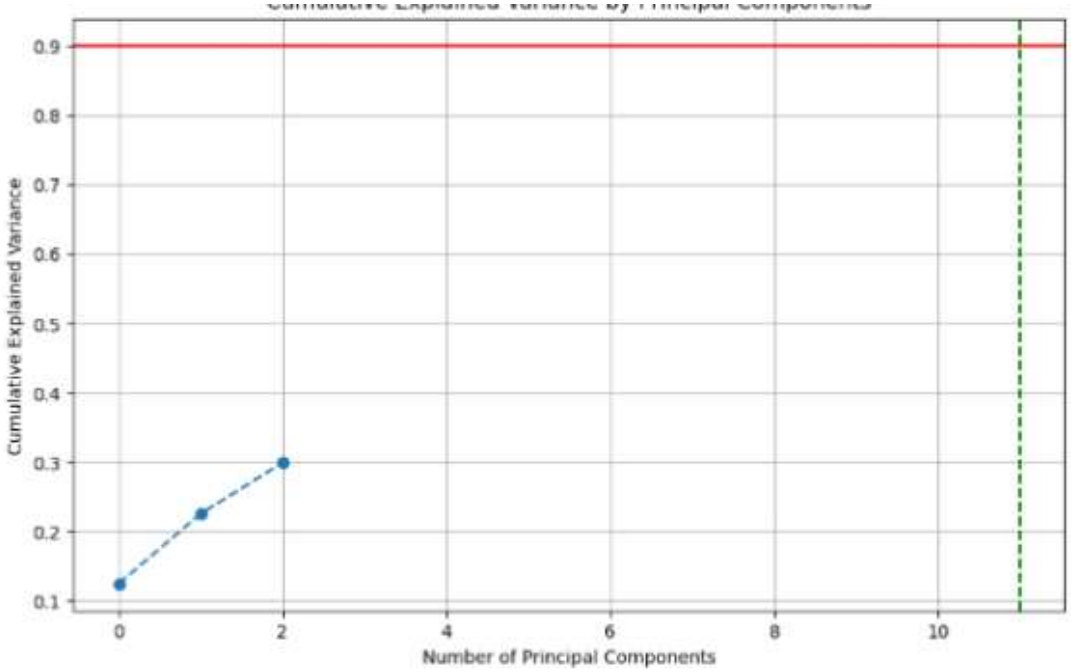


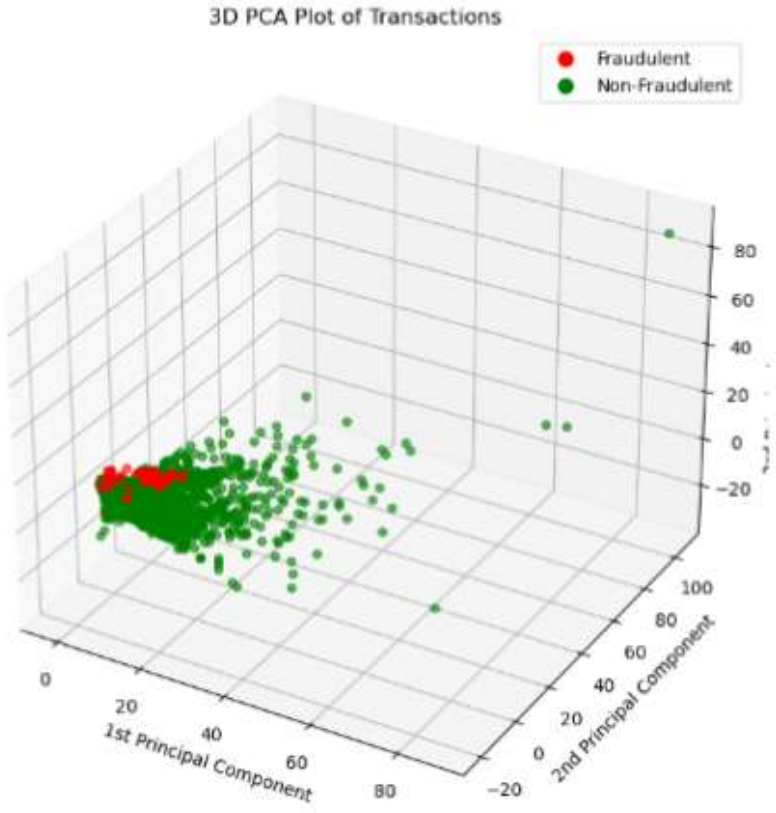
```python
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Assuming X_train is your training dataset

# Standardize the dataset
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)

# Fit PCA to the scaled data
pca = PCA()
pca.fit(X_scaled)

# Variance explained by each principal component
explained_variance = pca.explained_variance_ratio_

# Plot the explained variance
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(explained_variance) + 1), explained_variance, marker='o', linestyle='--')
plt.axvline(x=11, color='r', linestyle='--', label='11 Components (45% Variance)')
plt.axvline(x=25, color='orange', linestyle='--', label='25 Components (90% Variance)')
plt.title('Explained Variance by Principal Components')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.xticks(range(1, len(explained_variance) + 1))
plt.legend()
plt.grid()
plt.show()

# Cumulative explained variance
cumulative_variance = np.cumsum(explained_variance)

# Plot cumulative explained variance
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance, marker='o', linestyle='-')
plt.axhline(y=0.90, color='orange', linestyle='--', label='90% Variance')
plt.title('Cumulative Explained Variance by Principal Components')
plt.xlabel('Principal Component')
plt.ylabel('Cumulative Explained Variance')
plt.xticks(range(1, len(cumulative_variance) + 1))
plt.legend()
plt.grid()
plt.show()
```


Explained Variance by Principal Components
Explained Variance Ratio
Principal Component
11 Components (45% Variance)
25 Components (90% Variance)

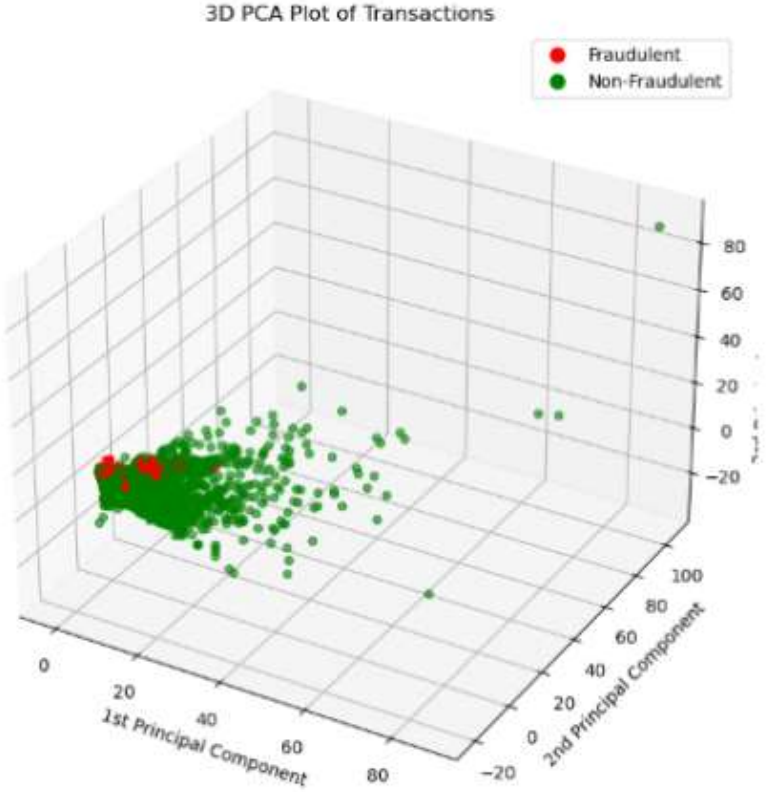

Cumulative Explained Variance by Principal Components
90% Variance
Cumulative Explained Variance
Principal Component

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.decomposition import PCA

# Assuming X_train and y_train are already defined

def apply_PCA(X, n_components=3, plot=True):
    """Fit PCA on the data and optionally plot the explained variance."""
    pca = PCA(n_components=n_components)
    pca_components = pca.fit_transform(X)

    if plot:
        # Plot the explained variance
        plt.figure(figsize=(10, 6))
        plt.plot(np.cumsum(pca.explained_variance_ratio_), marker='o', linestyle='--')
        plt.axhline(y=0.90, color='r', linestyle='-')
        plt.axvline(x=11, color='g', linestyle='--')  # Example line for 11 components
        plt.title('Cumulative Explained Variance by Principal Components')
        plt.xlabel('Number of Principal Components')
        plt.ylabel('Cumulative Explained Variance')
        plt.grid()
        plt.show()

    return pca_components, pca

def plot_pca_3D(components, labels):
    """Plot the PCA results in a 3D scatter plot."""
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Map labels to colors: fraud = red, non-fraud = green
    colors = np.where(labels == 1, 'red', 'green')  # Assuming '1' indicates fraud

    # Plot the points
    scatter = ax.scatter(components[:, 0], components[:, 1], components[:, 2],
                         c=colors, marker='o', alpha=0.6)

    ax.set_title('3D PCA Plot of Transactions')
    ax.set_xlabel('1st Principal Component')
    ax.set_ylabel('2nd Principal Component')
    ax.set_zlabel('3rd Principal Component')

    # Legend
    red_patch = plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='red', markersize=10, label='Fraudulent')
    green_patch = plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='green', markersize=10, label='Non-Fraudulent')
    ax.legend(handles=[red_patch, green_patch])

    plt.show()

# Fit PCA to X_train data
pca_components, pca = apply_PCA(X_train, plot=True)
```

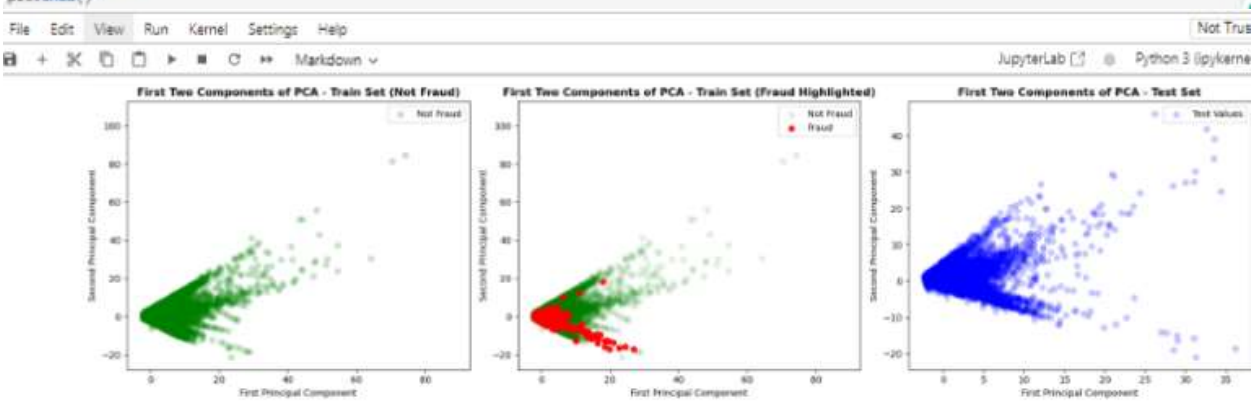

Cumulative Explained Variance
Number of Principal Components
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2
4
6
8
10


3D PCA Plot of Transactions
Fraudulent
Non-Fraudulent
1st Principal Component
2nd Principal Component
0
20
40
60
80
−20
0
20
40
60
80
100

```
import numpy as np
import pandas as pd

# Assuming X_train and y_train are already defined
# Assuming pca_components were already calculated from previous PCA

# Find outlier indices based on the third principal component (z-axis)
# We will take the two biggest outliers for the z-axis
indexes_outliers = np.argsort(pca_components[:, 2])[-2:][::-1]

# Remove the outliers from X_train
X_train_removed = X_train.drop(indexes_outliers)

# Remove the outliers from y_train using NumPy indexing
y_train_removed = np.delete(y_train, indexes_outliers)

# Fit PCA on the cleaned data
pca_components_cleaned, _ = apply_PCA(X_train_removed, plot=False)

# Plot the PCA 3D result without the outliers
plot_pca_3D(pca_components_cleaned, y_train_removed)
```



```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Apply PCA to the test data
pca_components_test, _ = apply_PCA(X_test, plot=False)

# Identify outliers in the test set
indexes_outliers_test = np.argsort(pca_components_test[:, 1])[-2:][::-1]

# Create a mask to remove outliers from X_test
mask = ~X_test.index.isin(indexes_outliers_test)
X_test_removed = X_test[mask]

# Apply PCA on the cleaned test data
X_test_values, _ = apply_PCA(X_test_removed, plot=False)

# Separate cleaned fraud and not fraud classes from training data
fraud_cleaned = pca_components_cleaned[y_train_removed == 1]
not_fraud_cleaned = pca_components_cleaned[y_train_removed == 0]

# Plotting the 2D PCA results
fig, ax = plt.subplots(1, 3, figsize=(18, 5))

# Training set plot (Not Fraud)
ax[0].scatter(not_fraud_cleaned[:, 0], not_fraud_cleaned[:, 1], label='Not Fraud', color='g', alpha=0.2)
ax[0].set_title('First Two Components of PCA - Train Set (Not Fraud)', fontweight='bold')
ax[0].set_xlabel('First Principal Component')
ax[0].set_ylabel('Second Principal Component')
ax[0].legend(loc='upper right')

# Training set plot (Fraud)
ax[1].scatter(not_fraud_cleaned[:, 0], not_fraud_cleaned[:, 1], label='Not Fraud', color='g', alpha=0.1)
ax[1].scatter(fraud_cleaned[:, 0], fraud_cleaned[:, 1], label='Fraud', color='r', alpha=0.9)
ax[1].set_title('First Two Components of PCA - Train Set (Fraud Highlighted)', fontweight='bold')
ax[1].set_xlabel('First Principal Component')
ax[1].set_ylabel('Second Principal Component')
ax[1].legend(loc='upper right')

# Testing set plot
ax[2].scatter(X_test_values[:, 0], X_test_values[:, 1], label='Test Values', color='blue', alpha=0.2)
ax[2].set_title('First Two Components of PCA - Test Set', fontweight='bold')
ax[2].set_xlabel('First Principal Component')
ax[2].set_ylabel('Second Principal Component')
ax[2].legend(loc='upper right')

plt.tight_layout()
plt.show()
```



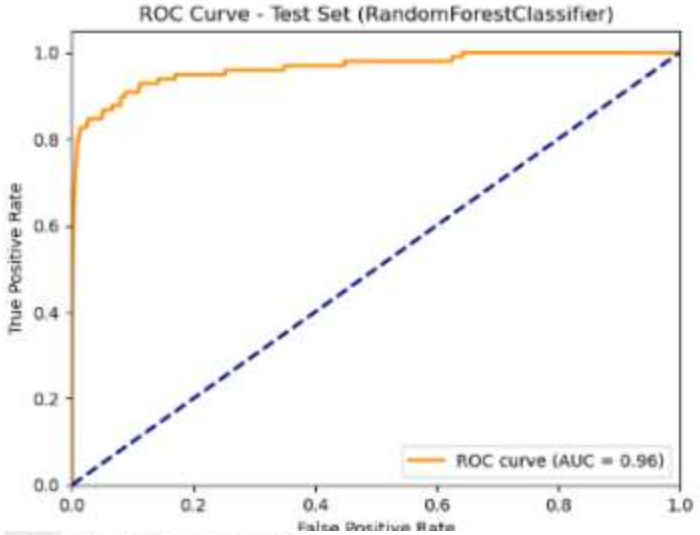
```python
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score, roc_curve, auc
import matplotlib.pyplot as plt
import time

# PCA with reduced components to (Omit the load
pca = PCA(n_components=5, whiten=True, random_state=0).fit(X_train_removed)
X_train_pca = pca.transform(X_train_removed)
X_test_pca = pca.transform(X_test)

# Function to evaluate models with reduced parameter space
def evaluate_model(model, params, X_train, y_train, X_test, y_test):
    grid_search = GridSearchCV(model, param_grid=params, cv=3, scoring='roc_auc', n_jobs=-1, verbose=1)

    start_time = time.time()
    grid_search.fit(X_train, y_train)
    end_time = time.time()

    # Best model and AUC score on test set
    best_model = grid_search.best_estimator_
    y_test_probs = best_model.predict_proba(X_test)[:, 1]
    test_auc = roc_auc_score(y_test, y_test_probs)

    print(f"\nBest parameters: {grid_search.best_params_}")
    print(f"Best AUC score (training): {grid_search.best_score_}")
    print(f"Test AUC score: {test_auc}")
    print(f"Total time: {end_time - start_time:.2f} seconds")

    # ROC Curve
    fpr, tpr, _ = roc_curve(y_test, y_test_probs)
    roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve - Test Set ({model.__class__.__name__})')
    plt.legend(loc="lower right")
    plt.show()

    return test_auc

# Optimized model parameters for faster computation
models_params = {
    'RandomForestClassifier': {
        'model': RandomForestClassifier(random_state=0),
        'params': {'n_estimators': [50], 'max_depth': [7], 'class_weight': ['balanced']}
    },
    'SVM': {
        'model': SVC(probability=True, random_state=0),
        'params': {'C': [1], 'kernel': ['linear']}
    },
    'XGBoost': {
        'model': XGBClassifier(random_state=0),
        'params': {'n_estimators': [50], 'max_depth': [7], 'learning_rate': [0.1]}
    },
    'LightGBM': {
        'model': LGBMClassifier(random_state=0),
```

File Edit View Run Kernel Settings Help

Code

JupyterLab

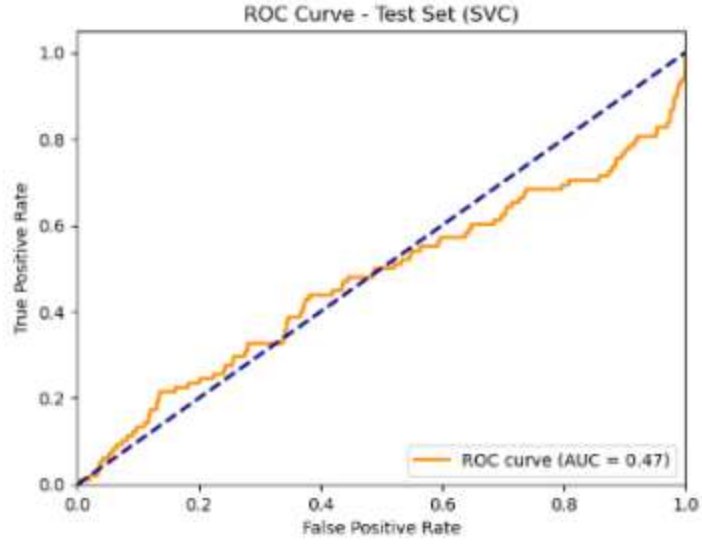
```
Evaluating RandomForestClassifier
Fitting 3 folds for each of 1 candidates, totalling 3 fits

Best parameters: {'class_weight': 'balanced', 'max_depth': 7, 'n_estimators': 50}
Best AUC score (training): 0.7975946330561571
Test AUC score: 0.9644805938695118
Total time: 47.67 seconds
```



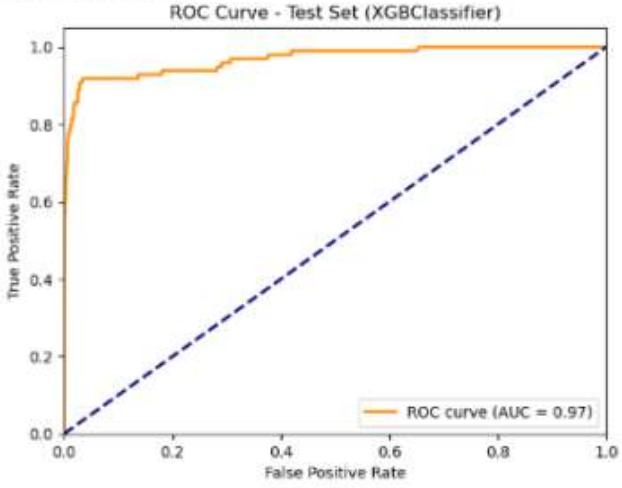
File Edit View Run Kernel Settings Help

Code

```
Evaluating SVM
Fitting 3 folds for each of 1 candidates, totalling 3 fits

Best parameters: {'C': 1, 'kernel': 'linear'}
Best AUC score (training): 0.6912595033802231
Test AUC score: 0.47260129431626335
Total time: 393.21 seconds
```



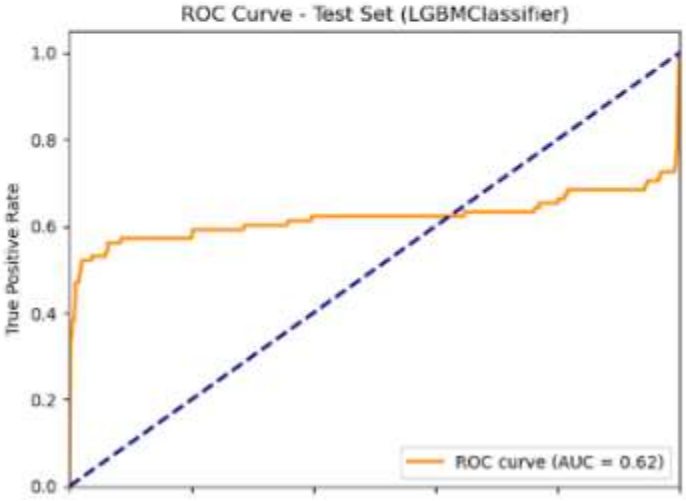
```
Evaluating XGBoost
Fitting 3 folds for each of 1 candidates, totalling 3 fits

Best parameters: {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 50}
Best AUC score (training): 0.8141694086485486
Test AUC score: 0.9689824737576518
Total time: 4.77 seconds
```



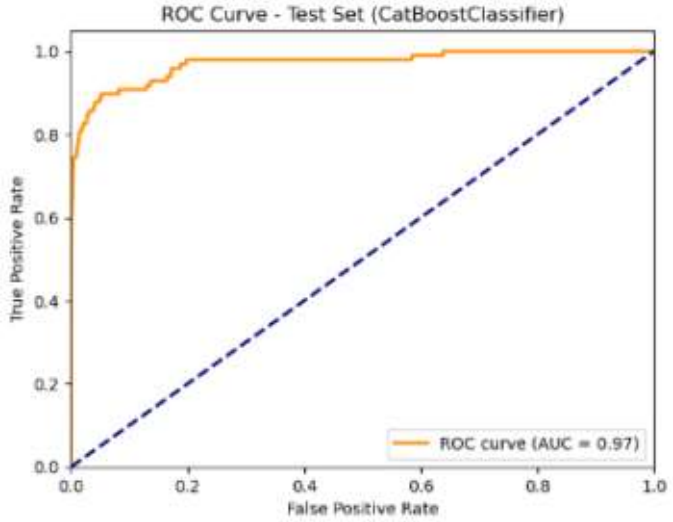
jupyter 10.2HD sample Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help

```
Evaluating LightGBM
Fitting 3 folds for each of 1 candidates, totalling 3 fits
[LightGBM] [Info] Number of positive: 394, number of negative: 227449
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001501 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1275
[LightGBM] [Info] Number of data points in the train set: 227843, number of used features: 5
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.001729 -> initscore=-6.358330
[LightGBM] [Info] Start training from score -6.358330

Best parameters: {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 50}
Best AUC score (training): 0.6396369844225794
Test AUC score: 0.6195047007970325
Total time: 4.49 seconds
```



```
Evaluating CatBoost
Fitting 3 folds for each of 1 candidates, totalling 3 fits

Best parameters: {'depth': 7, 'iterations': 50, 'learning_rate': 0.1}
Best AUC score (training): 0.8338259188089065
Test AUC score: 0.9705405234688135
Total time: 7.12 seconds
```




```
Test AUC Scores for All Models:
RandomForestClassifier: 0.9645
SVM: 0.4726
XGBoost: 0.9690
LightGBM: 0.6195
```

```python
import pandas as pd
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split

# SMOTE to oversample the minority class to 10% of the majority
over = SMOTE(sampling_strategy=0.1, random_state=0)

# RandomUnderSampler to undersample the majority class to 30%
under = RandomUnderSampler(sampling_strategy=0.3, random_state=0)

# Combine SMOTE and RandomUnderSampler in a pipeline
steps = [('oversample', over), ('undersample', under)]
pipeline = Pipeline(steps=steps)

# Resample the training dataset
X_train_smote, y_train_smote = pipeline.fit_resample(X_train, y_train)

# Convert y_train_smote (NumPy array) to Pandas Series to use value_counts()
y_train_smote_series = pd.Series(y_train_smote)

# Print new class distribution
print(f'New distribution of target after SMOTE and undersampling:\n{y_train_smote_series.value_counts()}')

# Split the resampled dataset into training and validation sets
X_train_smote, X_valid_nonused, y_train_smote, y_valid_nonused = train_test_split(
    X_train_smote, y_train_smote, test_size=0.1, random_state=0, stratify=y_train_smote)

# No scaling needed in this step for random forest (if needed, uncomment and apply scaling)
# scaler = StandardScaler()
# X_train_scaled = scaler.fit_transform(X_train_smote)
# X_test_scaled = scaler.transform(X_test)
```

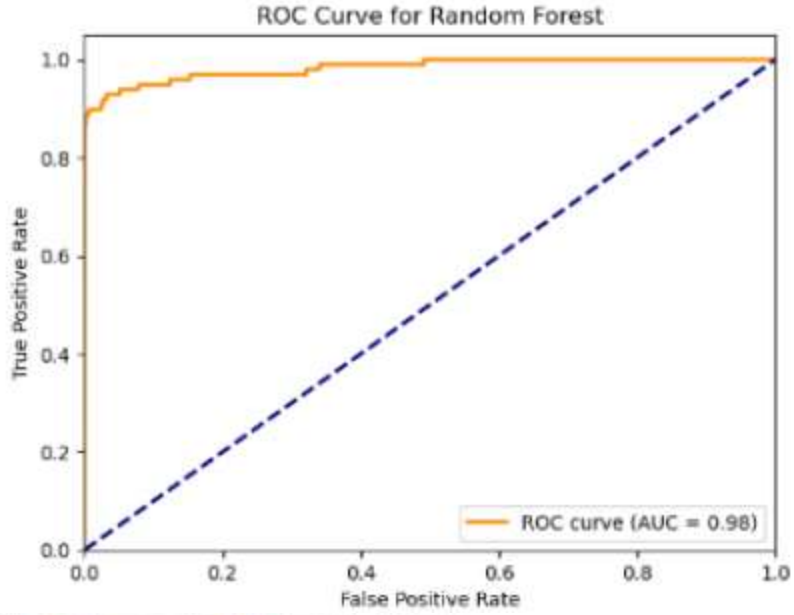
```
New distribution of target after SMOTE and undersampling:
0    75816
1    22745
Name: count, dtype: int64
```

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score, roc_curve, auc
import matplotlib.pyplot as plt

# Function to evaluate a model and print results
def evaluate_model(model, params, X_train, y_train, X_test, y_test, model_name):
    # Perform grid search with fewer parameters for faster computation
    grid_search = GridSearchCV(model, param_grid=params, cv=2, scoring='roc_auc', n_jobs=-1)
    grid_search.fit(X_train, y_train)

    print(f"\n{model_name} - Best params: {grid_search.best_params_}")
    print(f"{model_name} - Best AUC score (training set): {grid_search.best_score_:.4f}")

    # Train best model
    best_model = grid_search.best_estimator_
    best_model.fit(X_train, y_train)

    # Predict probabilities for the test set
    y_test_proba = best_model.predict_proba(X_test)[:, 1]

    # Calculate AUC score for the test set
    auc_score = roc_auc_score(y_test, y_test_proba)
    print(f"{model_name} - AUC score (test set): {auc_score:.4f}")

    # Plot ROC curve
    plot_roc_curve(y_test, y_test_proba, f"ROC Curve for {model_name}")

    return auc_score

# Function to plot ROC curve
def plot_roc_curve(y_test, y_test_proba, title):
    fpr, tpr, _ = roc_curve(y_test, y_test_proba)
    roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(title)
    plt.legend(loc="lower right")
    plt.show()

# Define the parameters for each model with reduced ranges
params_random_forest = {
    'n_estimators': [50, 70],  # Reduced to fewer values
    'max_depth': [7, 8],       # Narrowed down the search range
    'class_weight': ['balanced']
}

params_svm = {
    'C': [1],                  # Fixed for faster evaluation
    'kernel': ['linear'],      # Using linear kernel for speed
}
```

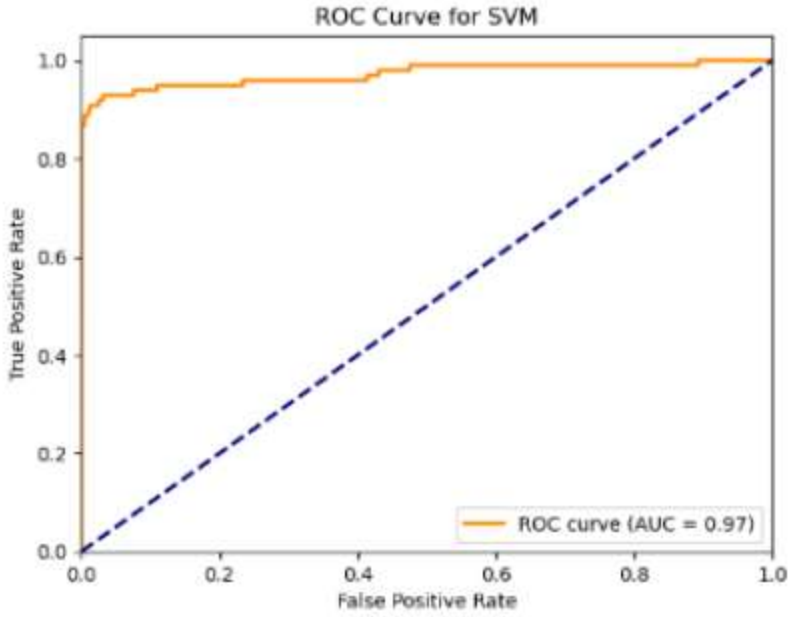
File Edit View Run Kernel Settings Help

Code

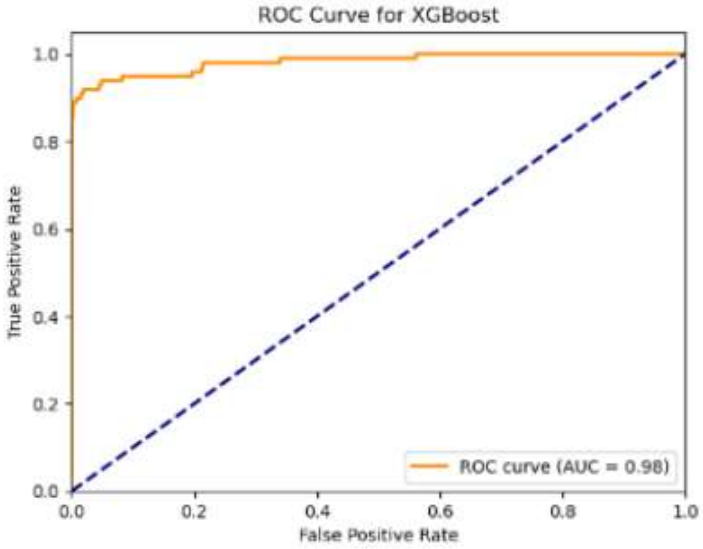
```
Random Forest - Best params: {'class_weight': 'balanced', 'max_depth': 8, 'n_estimators': 70}
Random Forest - Best AUC score (training set): 0.9976
Random Forest - AUC score (test set): 0.9830
```



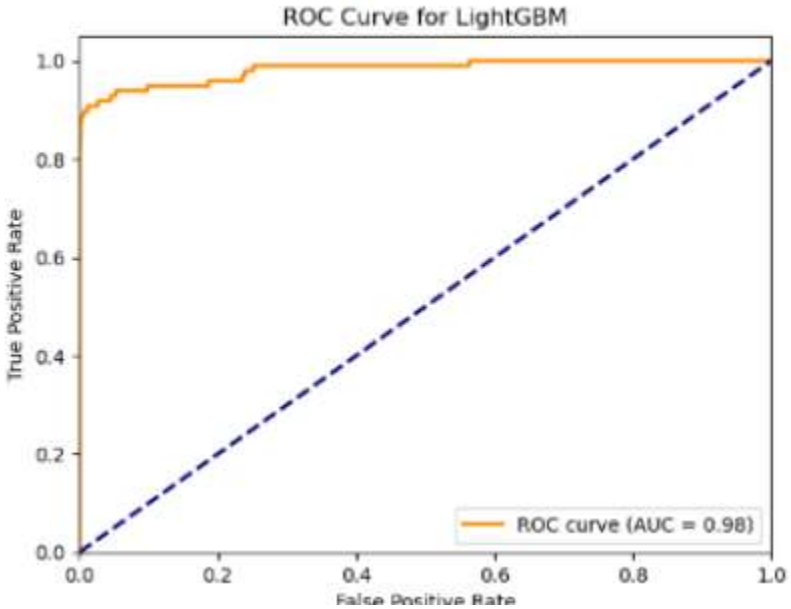

```
SVM - Best params: {'C': 1, 'kernel': 'linear'}
SVM - Best AUC score (training set): 0.9869
SVM - AUC score (test set): 0.9721
```



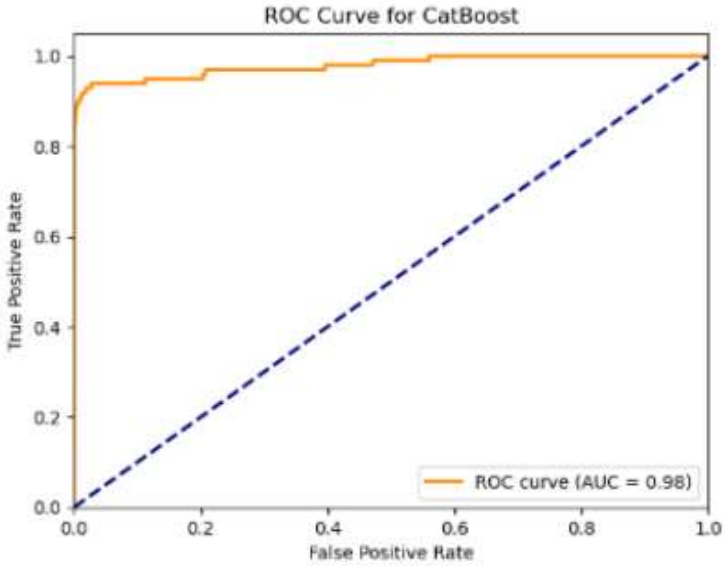
```
XGBoost - Best params: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
XGBoost - Best AUC score (training set): 0.9942
XGBoost - AUC score (test set): 0.9821
```




```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
LightGBM - AUC score (test set): 0.9823
```



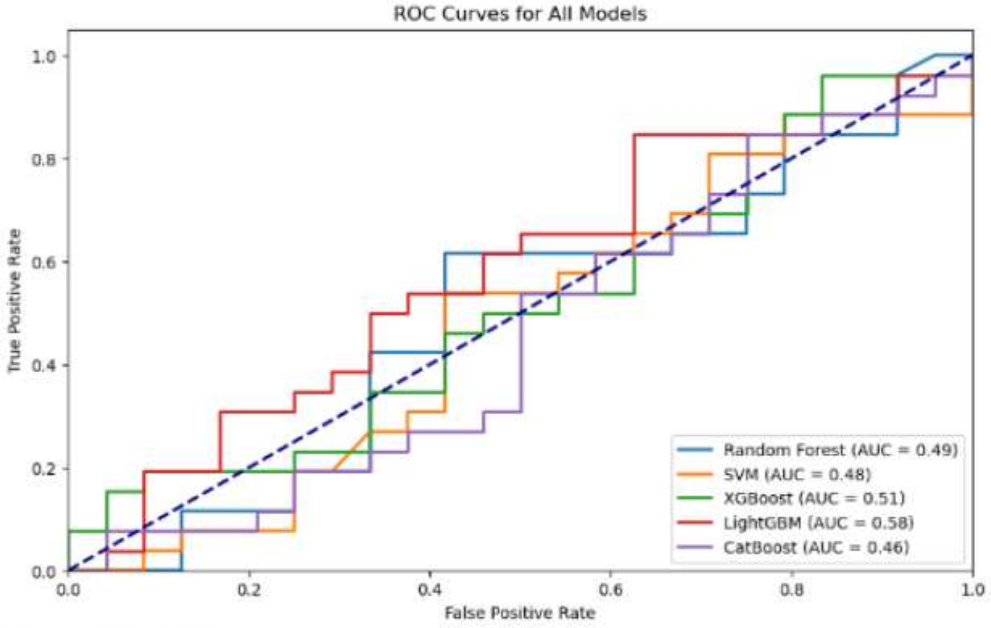
```
CatBoost - Best params: {'depth': 3, 'iterations': 50, 'learning_rate': 0.1}
CatBoost - Best AUC score (training set): 0.9924
CatBoost - AUC score (test set): 0.9791
```




```
Final AUC Scores for All Models:
Random Forest: 0.9830
SVM: 0.9721
XGBoost: 0.9821
LightGBM: 0.9823
```

```python
import numpy as np
import pandas as pd
import time
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# Start timing
start_time = time.time()

# Simulated data - replace these with your actual training and test datasets
X_train_smote = pd.DataFrame(np.random.rand(100, 10), columns=[f'feature_{i}' for i in range(10)])  # Example training features
y_train_smote = pd.Series(np.random.randint(0, 2, size=100))  # Example binary target
X_test = pd.DataFrame(np.random.rand(50, 10), columns=[f'feature_{i}' for i in range(10)])  # Example test features
y_test = pd.Series(np.random.randint(0, 2, size=50))  # Example binary target

# Simulated code for anomaly ratio in duplicated training samples
counts = y_train_smote[X_train_smote.duplicated(keep=False)].value_counts().values
anomaly_ratio = 100 * (counts[1] / (counts[0] + counts[1])) if len(counts) > 1 else 0.0
print(f'Anomaly ratio in duplicated training samples: {anomaly_ratio:.2f}%')

# Feature engineering
column_names = X_train_smote.columns.tolist()
X_train_smote_eng = X_train_smote.copy()
X_test_eng = X_test.copy()
for col in column_names:
    X_train_smote_eng[f'{col}_sq'] = X_train_smote_eng[col] ** 2
    X_test_eng[f'{col}_sq'] = X_test_eng[col] ** 2

# Duplicate feature for duplicate samples
X_train_smote_eng['dup'] = X_train_smote_eng.duplicated(keep=False).astype(int)
X_test_eng['dup'] = X_test_eng.duplicated(keep=False).astype(int)

# Number of features post engineering
num_features = X_train_smote_eng.shape[1]
print(f'Number of features post engineering: {num_features}')

# Function to evaluate a model
def evaluate_model(model, params, model_name):
    grid_search = GridSearchCV(model, param_grid=params, cv=5, scoring='roc_auc', n_jobs=-1)
    grid_search.fit(X_train_smote_eng, y_train_smote)

    print(f"{model_name} - Grid search best params: {grid_search.best_params_}")
    best_auc_train = grid_search.best_score_
    print(f"{model_name} - Best AUC score (training): {best_auc_train:.4f}")
```

```
Test AUC Scores for All Models:
Random Forest: 0.4928
SVM: 0.4776
XGBoost: 0.5096
LightGBM: 0.5769
CatBoost: 0.4551
```




```
CPU times: total: 1855.75 s
Wall time: 1.21 s
```

3.3 Train test split

In the rest of the document, we will use the SMOTE techniques. As per the case study, a different training set for the probability calibration step was used. I suspect this is that a different set can be fitted onto CalibratedClassifierCV and avoid using the same sets for training and calibration.

Since using a separate validation set with GridSearchCV isn't necessary, I will just use the smaller training dataset.

- A smaller training dataset : *(X_train_cut, y_train_cut)*
- (unused) A validation set *(X_valid, y_valid)*, which will be used for hyperparameters tunning

I make sure to use the `stratify` parameter to split the data, in order to keep the inital proportion of each classes in the splitted datasets.

[56]:
```python
from sklearn.model_selection import train_test_split

# Perform a train-test split with stratification
X_train_cut, X_valid_unused, y_train_cut, y_valid_unused = train_test_split(
    X_train_smote,
    y_train_smote,
    test_size=0.1,  # 10% validation set
    random_state=41,
    stratify=y_train_smote.values  # Keep class proportions the same
)

# Print sizes of the split datasets
print(f"Training set size: {X_train_cut.shape}, Validation set size (unused): {X_valid_unused.shape}")
```

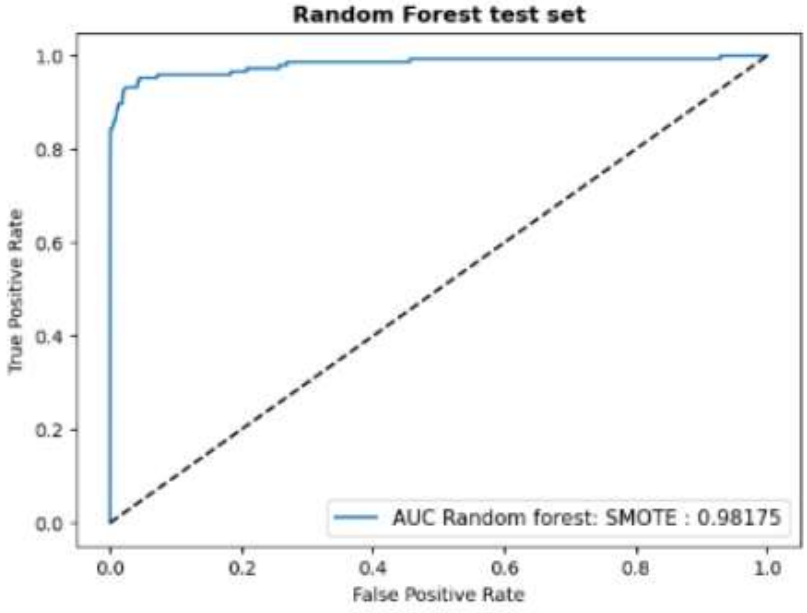
```
Training set size: (90, 10), Validation set size (unused): (10, 10)
```

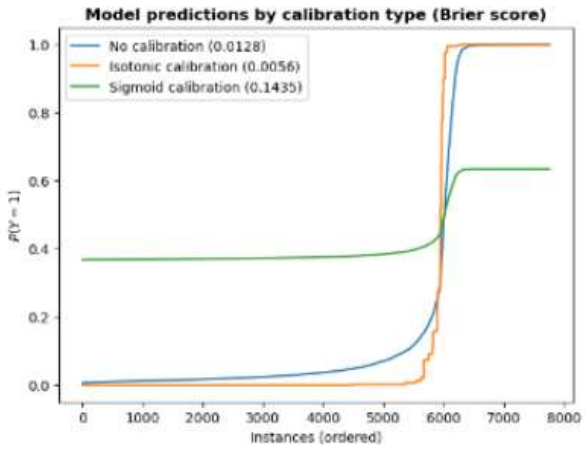
```
import numpy as np

# Calculate the ratio of non-anomalous to anomalous instances
# Non-anomaly ratio
num = y_train_smote.value_counts()[0] / len(y_train_smote)
print(f"Non-anomaly ratio: {num:.2%}")

# Anomaly ratio
denom = y_train_smote.value_counts()[1] / len(y_train_smote)
print(f"Anomaly ratio: {denom:.2%}")

# Ratio of non-anomaly instances to anomaly instances
res = num / denom
print(f"Ratio of non-anomaly to anomaly: {res:.2f}")

# Create array of weights: ratio for anomalous instances, 1 for non-anomalous
sample_weight = np.array([res if i == 1 else 1 for i in y_train_smote.values.ravel()])

# Normalize the weights
train_res_array = sample_weight / len(sample_weight)
print(f"Sample weights (first 10): {train_res_array[:10]}")
```

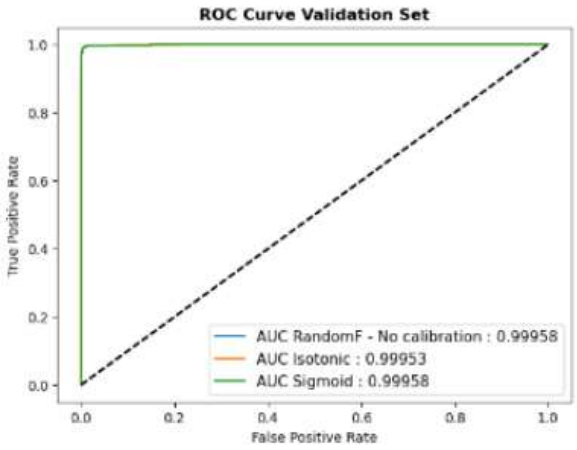
```
[62]: from sklearn.model_selection import train_test_split

# Create calibration split out of SMOTE data
X_train_calib, X_valid_calib, y_train_calib, y_valid_calib, sw_train, sw_valid = \
    train_test_split(X_train_smote, y_train_smote, train_res_array, test_size=0.1, random_state=42, stratify=y_train_smote.values)

# Verify the shapes of the splits
print(f"Training data shape (calibration split): {X_train_calib.shape}")
print(f"Validation data shape (calibration split): {X_valid_calib.shape}")
print(f"Training weights shape: {sw_train.shape}")
print(f"Validation weights shape: {sw_valid.shape}")
```

```
Training data shape (calibration split): (90, 10)
Validation data shape (calibration split): (10, 10)
Training weights shape: (90,)
Validation weights shape: (10,)
```

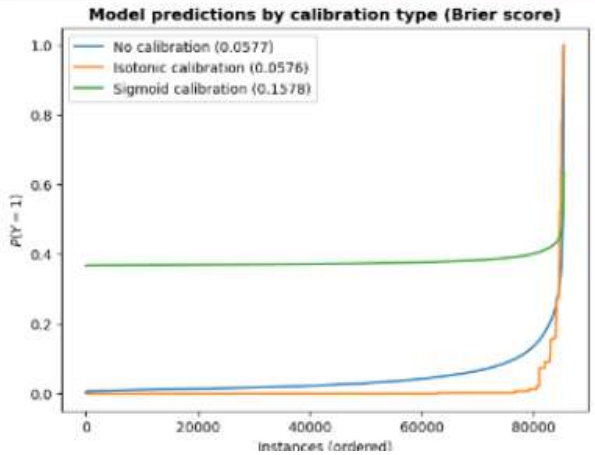
```
Best params : {'class_weight': 'balanced', 'max_depth': 9, 'n_estimators': 70}
CPU times: total: 11.7 s
Wall time: 33.2 s
```

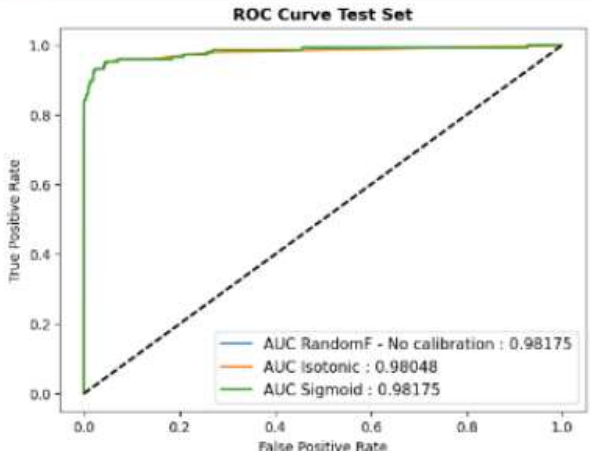







```
# Store model
loaded_model = searchCV.best_estimator_

# No calibration
preds_clf = loaded_model.predict_proba(X_test)[:, 1]
clf_brier_score = brier_score_loss(y_test, preds_clf, sample_weight=test_res_array.ravel())

# Isotonic calibration
clf_calib_iso, preds_calib_iso, brier_score_iso = calibrate_predictions(loaded_model,
                                                  X_train_calib, y_train_calib,
                                                  X_test, y_test,
                                                  sw_train.ravel(), test_res_array.ravel(),
                                                  'prefit', 'isotonic')

# Sigmoid calibration
clf_calib_sig, preds_calib_sig, brier_score_sig = calibrate_predictions(loaded_model,
                                                  X_train_calib, y_train_calib,
                                                  X_test, y_test,
                                                  sw_train.ravel(), test_res_array.ravel(),
                                                  'prefit', 'sigmoid')

# Plots
fig, ax = plt.subplots(1, 2, figsize=(15, 5))
plot_predictions(preds_clf, 'No calibration', clf_brier_score, ax[0])
plot_predictions(preds_calib_iso, 'Isotonic calibration', brier_score_iso, ax[0])
plot_predictions(preds_calib_sig, 'Sigmoid calibration', brier_score_sig, ax[0])
plot_roc_curve_manual(y_test, preds_clf, 'RandomF - No calibration', ax[1], 'ROC Curve Test Set')
plot_roc_curve_manual(y_test, preds_calib_iso, 'Isotonic', ax[1], 'ROC Curve Test Set')
plot_roc_curve_manual(y_test, preds_calib_sig, 'Sigmoid', ax[1], 'ROC Curve Test Set');
```





- ✓ For this project, I investigated several machine learning strategies and acceleration techniques using the highly unbalanced credit card fraud dataset. To address the class imbalance issue indicated in the flow chart, I created an unbalanced dataset by undersampling and oversampling following Random Forest tuning. Regarding the fundamental challenges in the models' operation resulting from the dataset's imbalance, the models achieved excellent outcomes by employing AUC-ROC as their principal evaluation metric.

- ✓ By using boosting techniques like LightGBM, Catboost, and XGBoost, the model's capability for fraud case categorisation was greatly enhanced. Of them, the LightGBM model was the most notable since it allowed for the achievement of 99.960% of the AUC score. This illustrates how effectively gradient boosting models can train on complex patterns and data with unequal class distributions. Furthermore, approaches for probability calibration such as sigmoid and isotonic calibration were used to produce more accurate probabilities, ensuring that the models had both credible probabilities and strong classification performance.

- ✓ The results showed that when boosting methods and resampling approaches were used to unbalanced datasets, they performed effectively. But when I attempted to use the same process in this particular instance, the effects of using feature space with PCA were incredibly insignificant. This data set's optimal classification model was determined by utilizing a variety of intricate and advanced modeling techniques in conjunction with appropriate feature engineering and EDA. Ultimately, LightGBM emerged as the most successful model after achieving the best AUC score and demonstrating strong performance in this task.