

# Personality Prediction Test

## Milestone: Performance Evaluation and Interpretation

### Group 16

Gouru Karthikeya Reddy

Giri Manohar Vemula

857-313-5329

857-832-1320

[reddy.go@northeastern.edu](mailto:reddy.go@northeastern.edu)

[vemula.gi@northeastern.edu](mailto:vemula.gi@northeastern.edu)

Percentage of Effort Contributed by Student 1: 50

Percentage of Effort Contributed by Student 2: 50

Signature of Student 1: Gouru Karthikeya Reddy

Signature of Student 2: Giri Manohar Vemula

Submission Date: 04-23-2023

## **Executive Summary:**

Determining which of the five primary personality types the respondent most closely aligns with in order to categorize them. Taking a personality test can assist you in selecting a career path by providing you with greater insight into how you respond to a variety of scenarios. The process of categorization becomes more difficult to do as the number of predictive features available for analysis grows. In order to simplify the dataset and take into account its suggested feature columns, we make use of a technique called Principal Component Analysis (PCA), which is a dimension reduction method. Following this, a solution should be proposed that models and classifies the data using machine learning techniques such as Logistic Regression, Decision Tree Classifier, Neural network, KNN Classifier, Naive Bayes, and Discriminant Analysis. This solution should integrate these machine learning approaches in both PCA and dataset prediction characteristics. A model that has been adequately trained and generalized will have the ability to predict the class of participants with a level of accuracy that is acceptable. Additionally, evaluation metrics such as Error Analysis, Lift Charts and ROC curves will be used to test the accuracy and measure of the models. The purpose of this research is to determine whether or not a phishing website can be recognized by its description and the essential characteristics it possesses.

## **I. Background and Introduction**

### **Background**

Career professionals and psychologists use this information in personality career tests for recruitment and candidate assessment. Accurate personality estimation is delicate and can be falsified to some extent by a seeker, as I did. Given that employment is frequently associated with significant fiscal and social benefits, job campaigners are incentivized to portray themselves in a way that best reflects the characteristics of an ideal seeker. This can lead campaigners to mask their true personality and result in inaccurate test assessments, particularly when carried out under formal surroundings. It also helps with an individual's behavior and preferences and provides personalized visualizations and could even suggest better music recommendations. In discrepancy, the assessment of personality in relaxed or informal surroundings may lead to further representative estimates of an existent's personality. The concept behind the test creates varied scales throughout an inventory and alternates between increment and decrement item scores, which encourages respondents to pay careful attention to the content of items, hence increasing the probability of entirely valid responses.

### **The Problem**

The assessment consists of fifty items that you must score on a scale of five points (like 1=Disagree, 3=Neutral, and 5=Agree) according to how accurate one is. Typically, it takes between 3 and 8 minutes to finish. Five personality characteristics such as extraversion, neuroticism, agreeableness, conscientiousness, and openness to experience respond to the majority of a person's personality questions. The big five are not connected with a specific exam; other methods have been created to assess them. Using the IPIP scoring key approach, we discover that res that are scored on a scale are divided into "+keyed" and "-keyed"

categories. Requires visualization of the personality tests of the participants in the data. The prerequisite to perform data pre-processing which is to clean the unstructured data by removing null values and continuing with clustering using machine learning algorithms which may be a centroid-based, distance-based algorithm, calculation of the distances to assign a point to a cluster. Furthermore, we need to use supervised machine learning, which is a technique that seeks to derive a collection of low-dimensional characteristics from a much larger set while still attempting to preserve as much variance as is humanly possible.

### **The goal of this study**

The purpose of the analysis is to properly recognize a class using a subset of the available feature columns.

### **Our Solution**

We make use of a technique called Principal Component Analysis (PCA), which is a dimension reduction method. Following this, a solution should be proposed that models and classifies the data using machine learning techniques such as Logistic Regression, Decision Tree, KNN Classifier, Naive Bayes, Neural network, and Linear Discriminant Analysis. This solution should integrate these machine learning approaches in both PCA and dataset prediction characteristics. A model that has been adequately trained and generalized will have the ability to predict the class of participants with a level of accuracy that is acceptable.

## **II. Data Exploration and Visualization**

### **Data Description**

This dataset comprises 964,573 online survey primary research responses conducted by Open Psychometrics. The personality test was developed using the International Personality Item Pool (IPIP)'s "Big-Five Factor Markers". Participants were reminded at the beginning of the examination that their responses would be recorded and used for research; at the end of the examination, they were asked to indicate their consent with having their responses recorded and used for research. On a single page, the following elements were displayed and assessed using radio buttons on a five-point scale. Overall, the dataset contains 964573 records and 112 features.

The EXT.. falls under 'Extroversion' category which is 'E' class in target, similarly 'Neuroticism', 'N': 'EST', 'Openness', 'O': 'OPN', 'Conscientiousness', 'C': 'CSN', 'Agreeableness', 'A' : 'AGR' . The sequence on the page follows EXT1, AGR3, CSN6, EST9, OPN1, EXT2, and so on. EXT1 I am the life of the party..... EXT10 I am quiet around strangers. EXT1 - EXT10 EST1 I get stressed out easily..... EST10 I often feel blue. EST1 - EST10 AGR1 I feel little concern for others..... AGR10 I make people feel at ease. AGR1 - AGR10 CSN1 I am always prepared..... CSN10 I am exacting in my work. CSN1 - CSN10 OPN1 I have a rich vocabulary..... OPN10 I am full of ideas. To understand the distribution of data, we analyzed the distributions of several attributes.

Milliseconds are used to measure how long an individual spends thinking about each question. These are the several variables whose names round out the sentence in \_E. EXT1\_E to EXT10\_E, EST1\_E to EST10\_E, AGR1\_E to AGR10\_E, CSN1\_E to CSN10\_E and OPN1 to OPN10. The dataset contains several features related to the behavior of users during a survey. These features include the ‘dateload’ timestamp, the screen width and height in pixels which are ‘screenw’ and ‘screenh’ respectively, the ‘introelapse’ time (in seconds) spent on the landing page, the ‘testelapse’ time (in seconds) spent on the page with survey questions, and the ‘endelapse’ time (in seconds) spent on the finalization page. The user's nation, as well as the user's estimated latitude and longitude, ‘lat\_appx\_lots\_of\_err’ and ‘long\_appx\_lots\_of\_err’, respectively, are also included in the dataset. Additionally, the IPC value, which reflects the number of records from the user's IP address in the dataset, is included in the dataset. In conclusion. Mainly, the dataset has a target column, which reflects the characteristic that received the highest possible score on the test, which is our response/target variable.

	Unnamed: 0	id	EXT1	EXT2	EXT3	EXT4	EXT5	EXT6	EXT7	EXT8	...	screenh	introelapse
count	964573.000000	9.645730e+05	962863.000000	962863.000000	962863.000000	962863.000000	962863.000000	962863.000000	962863.000000	962863.000000	...	962611.000000	9.626110e+05
unique	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
top	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
freq	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
first	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
last	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
mean	482286.000000	5.076880e+05	2.647965	2.772884	3.288141	3.140707	3.277330	2.400889	2.771749	3.414710	...	826.316675	9.574240e+02
std	278448.384939	2.930907e+05	1.264292	1.323763	1.215109	1.237310	1.277449	1.225707	1.400459	1.271566	...	180.259150	5.191203e+04
min	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000e+00
25%	241143.000000	2.539070e+05	1.500000	2.000000	2.000000	2.000000	2.000000	1.000000	2.000000	2.000000	...	720.000000	5.000000e+00
50%	482286.000000	5.076710e+05	3.000000	3.000000	3.000000	3.000000	3.000000	2.000000	3.000000	4.000000	...	768.000000	1.000000e+01
75%	723429.000000	7.614860e+05	4.000000	4.000000	4.000000	4.000000	4.000000	3.000000	4.000000	4.000000	...	900.000000	3.000000e+01
max	964572.000000	1.015340e+06	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	...	8802.000000	2.944307e+07

13 rows × 113 columns

Figure 1: Data Summary

## Exploratory Data Analysis and Data Visualization

There are four distinct data types, which are referred to as int64(3), float32(100), float64(4), datetime64[ns](1), and object(4), and the count of each type is mentioned in brackets. These data types need to be formatted or transformed into a format that is legible.

Analyze the statistics of the dataset, then remove the null values and inappropriate columns. Creating new columns based on existing columns and dropping them to reduce data size.

Using the selected feature columns for performance validation.

Distribution analysis of the participants' response and target variable for balance of target variable classes, and we observe O & A classes exhibit a significant imbalance, this can lead to biased predictions. This issue will be solved before training the models.

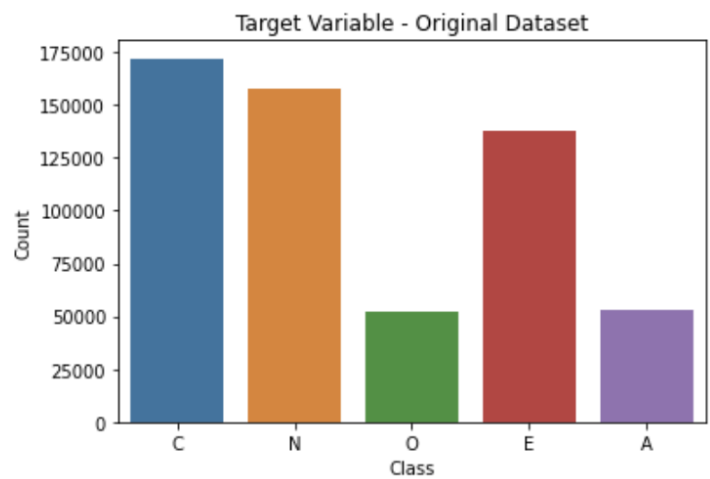
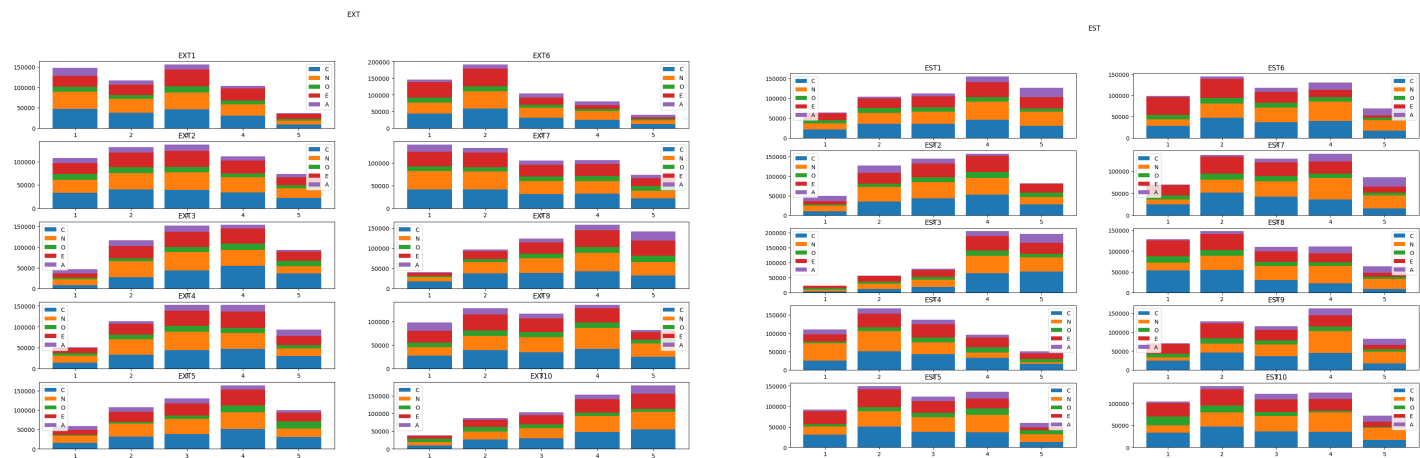


Figure 2: Distribution analysis of the target variable.



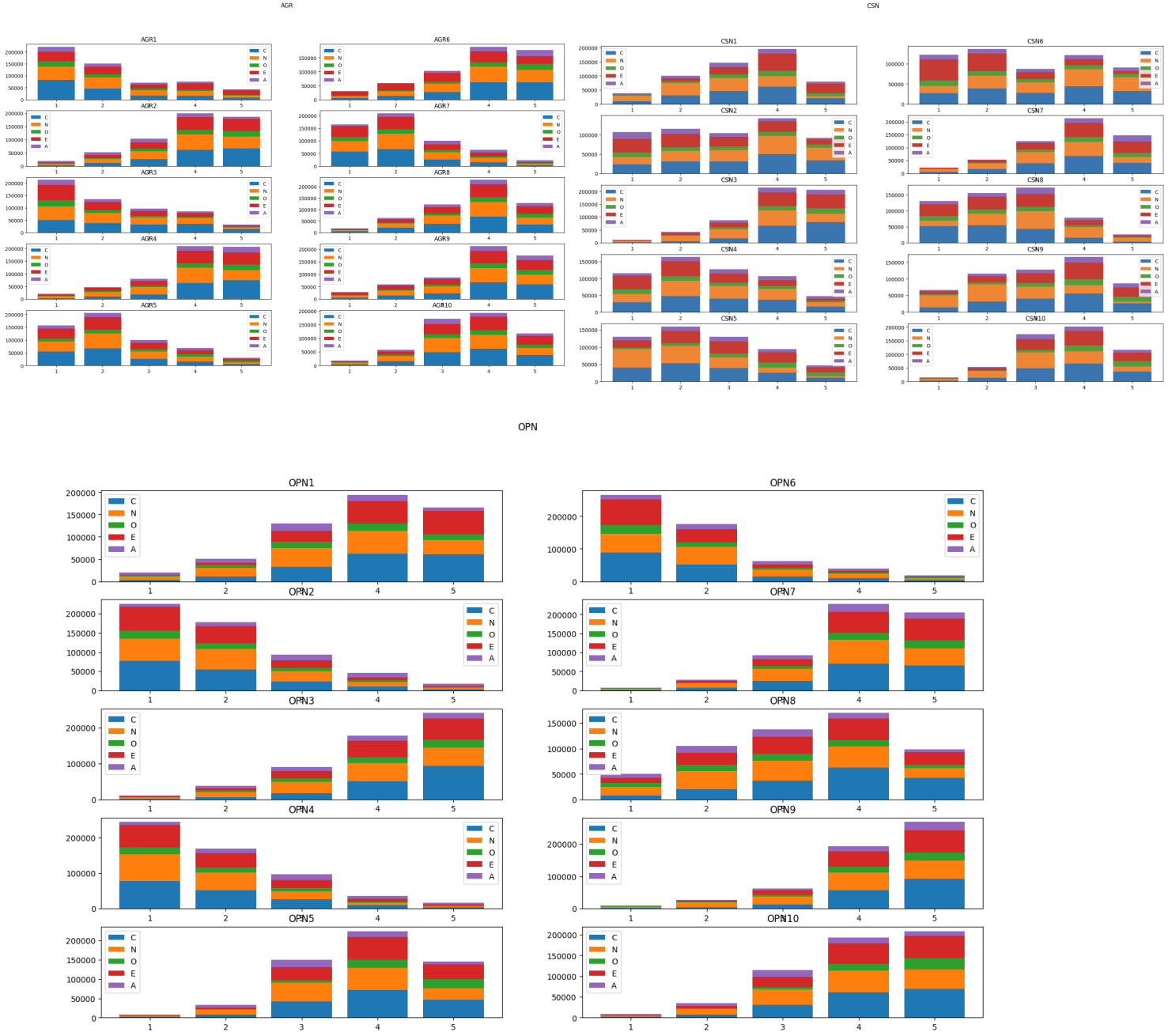


Figure 3: Distribution analysis of the question feedback.

Summary statistics of the dataset were produced, including the mean, the standard deviation, the minimum value, the maximum value, the median, the length (number of observations), and the number of missing values. Although the minimum and maximum values demonstrate the range of the variable, the mean value and standard deviation of the variable's values provide a knowledge of the central tendency and dispersion of the variable's values, respectively. The minimum and maximum values illustrate the range of the variable.

	mean	sd	min	max	median	length	miss.val
<b>AGR1</b>	2.232008	1.303506	1.0	5.0	2.0	561204	0
<b>AGR10</b>	3.597277	1.035181	1.0	5.0	4.0	561204	0
<b>AGR2</b>	3.856450	1.086239	1.0	5.0	4.0	561204	0
<b>AGR3</b>	2.267879	1.264163	1.0	5.0	2.0	561204	0
<b>AGR4</b>	3.947841	1.080500	1.0	5.0	4.0	561204	0
<b>AGR5</b>	2.301817	1.155668	1.0	5.0	2.0	561204	0
<b>AGR6</b>	3.758257	1.166108	1.0	5.0	4.0	561204	0
<b>AGR7</b>	2.233012	1.112323	1.0	5.0	2.0	561204	0
<b>AGR8</b>	3.685909	1.046487	1.0	5.0	4.0	561204	0
<b>AGR9</b>	3.792477	1.140049	1.0	5.0	4.0	561204	0
<b>CSN1</b>	3.320381	1.124064	1.0	5.0	3.0	561204	0

Length of the variable is just the observations count that are contained within it, but the number of missing values is a count that indicates how many observations are absent. These summary statistics may help with further analysis as well as provide insights into the features of the variable being studied.

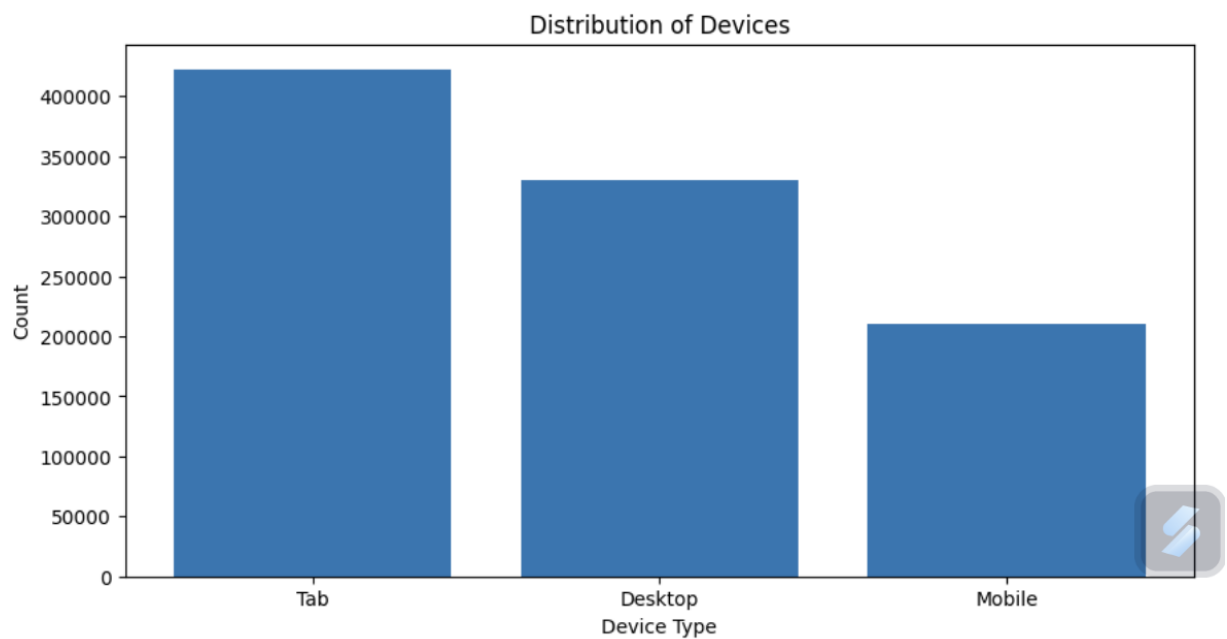


Figure 5: Distribution of Devices

### III. Data Preparation and Preprocessing

#### Data Summary


Our dataset consists of numerical attributes and a categorical target attribute. The “target” column represents the response variable which is a single categorical variable.

#### Variable Selection

Since our dataset comprises many correlated attributes, it becomes a necessity to select only linearly independent attributes. However, selecting and discarding them requires domain knowledge in cyber security. To tackle this, we perform principal component analysis on the data to reduce it into independent and information dense components.

#### Preprocessing and Dimensionality Reduction

As explained in the above part, we changed the “dataload” column to datetime format and stored it in the “date\_column” column. Converted the “lat\_appx\_lots\_of\_err” and “long\_appx\_lots\_of\_err” to numeric data type, “country” and “target” to string datatype. Dropped the first two columns which have no order and necessary for validation. “Country” column provides alpha code-2 of countries which we used to figure out the country names and store it in a new column. Using “screenw” and “screenh” columns which represent screen width and screen height of participant device dimensions respectively, we create two columns “screen\_size”, which is the diagonal screen size of device and based on values we set up “device\_type” column. “IPC” describes the count of records from IP address of the user in the data, an interesting observation we made was that the IP address count is more than “1”, so its value should be “1” to ignore duplicate data and so we filter it on condition of value equal to one. New “Time consumed” column which is summation of three main time columns “introelapse”, “testelapse” and “endelapse”.

	column_name	nullv_values	percent_nullv (%)	
Countries	Countries	18396	2.778886	
long_appx_lots_of_err	long_appx_lots_of_err	18256	2.757737	
lat_appx_lots_of_err	lat_appx_lots_of_err	18256	2.757737	
Time consumed	Time consumed	3618	0.546532	
device_type	device_type	2290	0.345926	
...	...	...	...	
CSN1	CSN1	2196	0.331726	
AGR10	AGR10	2196	0.331726	
EXT5_E	EXT5_E	2196	0.331726	
target	target	0	0.000000	
date_column	date_column	0	0.000000	

108 rows × 3 columns



In continuation to data cleaning, we observe the null and duplicate values from modified data and remove them. We replaced a few missing values in the “Countries” column using "lat\_appx\_lots\_of\_err" ,"long\_appx\_lots\_of\_err" and replaced null with country names. The time columns are measured in milliseconds, converted to seconds format.

	EXT1_E	EXT2_E	EXT3_E	EXT4_E	EXT5_E	EXT6_E	EXT7_E	EXT8_E	EXT9_E	EXT10_E	...	OPN1_E	OPN2_E	OPN3_E	OPN4_E	OPN5_E	OPN6_E	OPN7_E	OPN8_E	OPN9_E	OPN10_E	
3	8.624	2.872	5.600	3.152	5.775	1.753	4.208	5.648	7.136	1.704	...	2.287	4.529	2.440	2.176	2.832	2.864	1.889	2.032	2.976	0.544	
5	6.104	6.800	9.608	4.903	7.128	4.103	6.194	12.332	6.184	4.375	...	2.261	9.867	8.887	3.440	2.176	4.490	4.361	2.264	3.617	1.784	
6	18.124	3.265	4.351	5.213	4.557	3.280	7.985	8.316	3.831	14.332	...	5.200	12.331	5.740	4.318	7.449	7.166	6.059	3.805	3.269	6.284	
9	415.090	6.274	5.648	6.534	2.266	2.662	5.214	4.602	4.116	2.524	...	3.504	5.696	3.018	9.388	8.514	8.082	13.366	3.024	4.224	2.870	
10	9.913	4.850	3.067	5.149	2.818	2.657	21.011	6.667	5.437	4.240	...	4.674	5.763	3.466	3.407	7.731	6.203	5.778	5.269	23.206	3.347	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
964568	3.199	1.369	2.831	2.084	9.359	2.681	4.443	2.213	5.905	1.483	...	3.141	7.862	2.022	1.929	1.935	3.038	3.419	2.222	5.454	1.204	
964569	5.796	2.856	3.753	2.467	2.366	2.851	5.017	7.421	2.905	2.044	...	8.089	3.115	2.253	3.089	2.281	2.212	0.969	3.269	4.595	1.236	
964570	7.829	2.525	1.822	1.311	1.739	2.213	2.768	3.026	2.471	1.754	...	1.729	2.615	1.853	2.451	0.655	1.927	2.135	2.547	1.184	1.495	
964571	38.281	3.046	5.615	2.685	3.667	3.367	34.798	4.067	3.184	4.614	...	2.836	3.494	2.450	3.016	2.770	4.235	3.020	4.233	2.921	2.382	
964572	15.417	3.635	3.336	6.967	2.350	4.817	4.085	3.355	3.219	2.288	...	4.591	7.287	1.814	3.838	2.137	2.609	2.714	5.991	2.647	1.851	

561307 rows × 50 columns

Figure 6: Dataframe of Participant Information

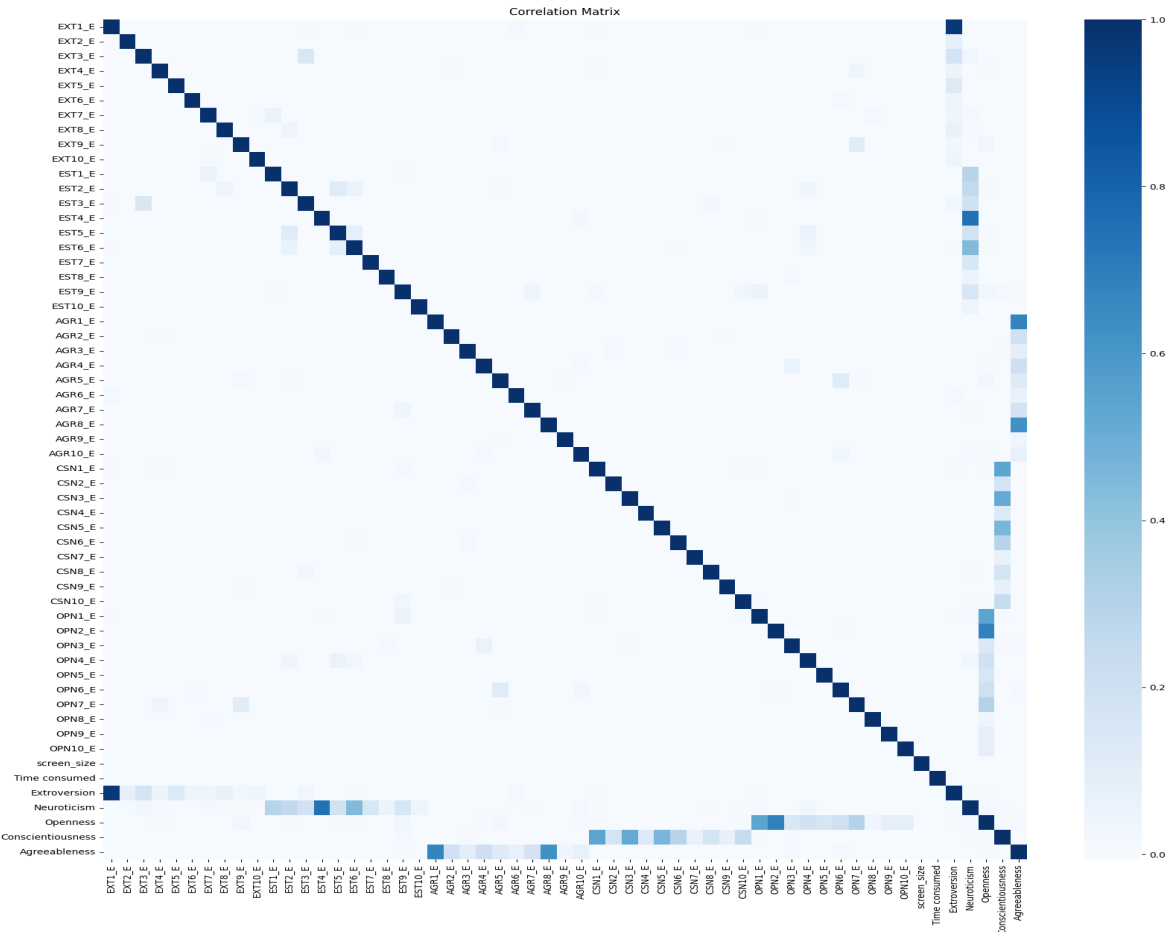


Figure 7: Correlation Matrix of Participant Information

## Feature Selection

We use SelectKBest method with the k parameter set to 60 from the sklearn.feature\_selection module for selecting the top k features with the highest scores, based on the given scoring function. We fit the SelectKBest object to the input data (in this case, the first 100 columns) and the target variable (the target column). This calculates the scores for each feature using the default scoring function, which is `f_classif` for classification tasks and `f_regression` for regression tasks. We select the top 50 as feature columns, which we got question response columns (EXT1 to OCN10) for performance evaluation, and plot the correlation matrix.

Correlation Matrix:

	EXT1	EXT2	EXT3	EXT4	EXT5	EXT6	EXT7	EXT8	EXT9	EXT10	...	OPN1	OPN2	OPN3	OPN4	OPN5	OPN6	OPN7	OPN8	OPN9	OPN10
EXT1	1.000000	-0.475246	0.474405	-0.534324	0.496543	-0.355203	0.593355	-0.409991	0.483543	-0.456134	...	0.041765	-0.023818	0.076133	-0.000983	0.189073	-0.082855	0.078462	0.017649	-0.078352	0.162405
EXT2	-0.475246	1.000000	-0.457367	0.573886	-0.558744	0.555816	-0.481445	0.410069	-0.399756	0.509202	...	-0.072370	0.049426	-0.055880	0.033896	-0.139015	0.090288	-0.060954	-0.046715	0.052520	-0.143104
EXT3	0.474405	-0.457367	1.000000	-0.491994	0.569931	-0.361695	0.552815	-0.342855	0.429745	-0.495428	...	0.027839	-0.053665	0.002036	-0.004531	0.155722	-0.054992	0.117485	-0.047842	-0.078529	0.110609
EXT4	-0.534324	0.573886	-0.491994	1.000000	-0.528001	0.465040	-0.516977	0.506175	-0.491063	0.554973	...	-0.057703	0.078867	0.003616	0.041320	-0.166026	0.076890	-0.088093	-0.024444	0.100013	-0.131341
EXT5	0.496543	-0.558744	0.569931	-0.528001	1.000000	-0.460895	0.612019	-0.371442	0.427922	-0.564293	...	0.086485	-0.060706	0.073707	-0.044428	0.203217	-0.103392	0.108830	0.040116	-0.010447	0.184858
EXT6	-0.355203	0.555816	-0.361695	0.465040	-0.460895	1.000000	-0.373024	0.336861	-0.339923	0.404933	...	-0.204754	0.194532	-0.132011	0.168970	-0.265633	0.194024	-0.164753	-0.153585	-0.059304	-0.279877
EXT7	0.593355	-0.481445	0.552815	-0.516977	0.612019	-0.373024	1.000000	-0.371910	0.439209	-0.551291	...	0.052948	-0.046157	0.057076	-0.023623	0.165192	-0.072141	0.088857	0.026882	-0.047525	0.157530
EXT8	-0.409991	0.410069	-0.342855	0.506175	-0.371442	0.336861	-0.371910	1.000000	-0.546106	0.415879	...	-0.057772	0.058202	-0.055582	0.048689	-0.147674	0.082969	-0.039322	-0.056051	0.071422	-0.128057
EXT9	0.483543	-0.399756	0.429745	-0.491063	0.427922	-0.339923	0.439209	-0.546106	1.000000	-0.403040	...	0.101509	-0.081390	0.091623	-0.056737	0.218471	-0.097468	0.123630	0.106736	-0.032319	0.190478
EXT10	-0.456134	0.509202	-0.495428	0.554973	-0.564293	0.404933	-0.551291	0.415879	-0.403040	1.000000	...	-0.067356	0.078568	-0.011491	0.039484	-0.144957	0.069131	-0.079320	-0.031822	0.102671	-0.119311
EST1	-0.108673	0.050517	-0.239545	0.146780	-0.126723	0.095798	-0.136812	0.098989	-0.145425	0.180542	...	-0.059365	0.169388	0.047322	0.071268	-0.138125	0.050176	-0.177418	-0.014764	0.080775	-0.080139
EST2	0.127048	-0.014271	0.277913	-0.072742	0.118175	-0.023406	0.137725	-0.042301	0.133419	-0.094775	...	0.013592	-0.075502	-0.004187	-0.022215	0.111663	-0.037378	0.128608	-0.035539	-0.056176	0.061715
EST3	-0.113428	0.050264	-0.197759	0.166766	-0.099497	0.074260	-0.128750	0.120228	-0.132438	0.193123	...	-0.042916	0.125716	0.101939	0.038348	-0.106194	0.016580	-0.118306	-0.004150	0.153070	-0.041347
EST4	0.115581	-0.072052	0.215564	-0.122446	0.125664	-0.050747	0.128826	-0.073814	0.112179	-0.130751	...	-0.033408	0.012322	-0.059264	0.076717	0.077037	0.025753	0.068264	-0.063834	0.012223	0.012223
EST5	-0.059595	0.035199	-0.166527	0.101210	-0.077973	0.108280	-0.086884	0.041707	-0.068933	0.112368	...	-0.091575	0.169560	0.024951	0.095733	-0.067716	0.062585	-0.144343	-0.032005	0.033473	-0.064693
EST6	-0.072309	0.016371	-0.206712	0.109252	-0.096019	0.088587	-0.107508	0.039539	-0.077576	0.140981	...	-0.081092	0.179428	0.050836	0.100124	-0.114522	0.059078	-0.160752	-0.015111	0.046830	-0.068875
EST7	-0.016189	0.009817	-0.184283	0.090901	-0.072265	0.071073	-0.052383	0.004875	-0.015425	0.119241	...	-0.065563	0.129694	0.103645	0.050647	-0.076009	0.012236	-0.110529	0.026085	0.065351	-0.007706
EST8	-0.034606	0.024821	-0.215749	0.103224	-0.093316	0.082718	-0.074757	0.007068	-0.034711	0.120094	...	-0.058692	0.131612	0.093297	0.055057	-0.087942	0.023849	-0.125662	0.049295	0.061520	-0.020197
EST9	-0.070436	0.031664	-0.233876	0.111572	-0.117202	0.085040	-0.122482	0.049824	-0.063130	0.159007	...	-0.041996	0.153135	0.035619	0.104805	-0.071366	0.062981	-0.114954	0.038100	0.042049	-0.050154
EST10	-0.188821	0.178295	-0.354406	0.254845	-0.230220	0.173508	-0.214995	0.130254	-0.157721	0.249678	...	0.020195	0.050245	0.091336	-0.024675	-0.126727	0.024041	-0.108918	0.098197	0.158010	-0.017767
AGR1	-0.041683	0.129915	-0.129978	0.089337	-0.120963	0.159801	-0.083650	-0.006342	0.016672	0.077311	...	-0.079542	0.083440	-0.033135	0.094911	-0.004913	0.060945	-0.013056	-0.024206	-0.109282	-0.036479
AGR2	0.273023	-0.300778	0.415195	-0.258367	0.385933	-0.290402	0.346896	-0.183959	0.220441	-0.259075	...	-0.065962	-0.055563	0.098952	-0.108421	0.097070	-0.094367	0.065574	0.000414	0.102142	0.135826
AGR3	0.058674	-0.041904	-0.091357	-0.001546	-0.026110	-0.002960	-0.005659	-0.070841	0.100167	0.025897	...	0.040519	0.037920	0.040621	0.054037	0.028532	0.032986	-0.009984	0.129219	-0.035826	0.037289
AGR4	0.095690	-0.129151	0.206119	-0.057698	0.185353	-0.118060	0.140123	-0.001873	0.014314	-0.065988	...	-0.008728	-0.008398	0.090380	-0.065254	0.000978	-0.074119	0.001210	-0.063592	0.137209	0.041884
AGR5	-0.120224	0.206972	-0.233367	0.155639	-0.231221	0.212759	-0.173208	0.082984	-0.059400	0.148498	...	-0.015704	0.050419	-0.041866	0.119552	-0.003094	0.073785	-0.004819	0.029416	-0.087868	-0.047697
AGR6	0.051266	-0.062050	0.106574	-0.000634	0.097946	-0.006966	0.069096	0.031547	-0.026016	0.004832	...	-0.077285	0.080811	0.064020	0.012428	-0.019320	-0.034706	-0.053166	-0.104993	0.090064	0.004244
AGR7	-0.229718	0.302504	-0.374340	0.267487	-0.341846	0.310388	-0.319609	0.173071	-0.164600	0.255505	...	-0.034323	0.070148	-0.065443	0.131607	-0.055543	0.127388	-0.027755	0.030003	-0.073801	-0.099442
AGR8	0.155579	-0.168492	0.262051	-0.121120	0.242483	-0.147509	0.203596	-0.014461	0.071333	-0.116383	...	-0.000431	-0.004734	0.051498	-0.030148	0.046448	-0.050405	0.050065	-0.036483	0.096411	0.069919
AGR9	0.134556	-0.144921	0.196123	-0.093428	0.202609	-0.132784	0.164999	-0.041652	0.074167	-0.087607	...	-0.007439	-0.016230	0.119571	-0.063689	0.040759	-0.099971	0.025519	-0.040732	0.147183	0.085827

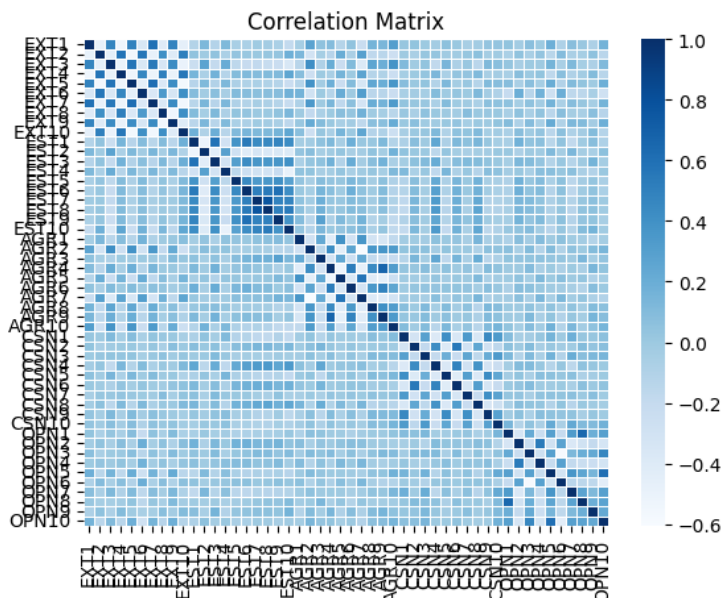


Figure 8 and 9: Correlation Matrix of Participant Responses and its Heatmap

It seems that a number of attributes are either highly correlated with one another in a favorable way or strongly associated with one another in a bad way. This is a dataset that has been cleaned from duplicacy, and made it appropriate for data mining techniques, it will need to go through the processes of cleaning and transformation.

## PCA :

It is possible to improve the effectiveness of machine learning algorithms by minimizing the danger of overfitting and the computational load of training models on big datasets. To do this, we use principal component analysis (PCA), which reduces the dimensionality of the predictors in a dataset.

We compute 43 numbers of components required to achieve 95% of the variance in the predictors. We are using the 43rd PCA component list columns as predictors for test-train evaluation.

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	...	PC34	PC35	PC36	PC37	PC38	PC39	PC40	PC41	PC42	PC43
Standard deviation	2.779	2.249	2.002	1.895	1.703	1.205	1.150	1.016	0.985	0.962	...	0.665	0.655	0.646	0.645	0.637	0.629	0.625	0.617	0.603	0.601
Proportion of variance	0.154	0.101	0.080	0.072	0.058	0.029	0.026	0.021	0.019	0.018	...	0.009	0.009	0.008	0.008	0.008	0.008	0.008	0.008	0.007	0.007
Cumulative proportion	0.154	0.256	0.336	0.407	0.465	0.495	0.521	0.542	0.561	0.580	...	0.884	0.893	0.901	0.910	0.918	0.926	0.933	0.941	0.948	0.956

3 rows × 43 columns

*Figure 10: Summary of 43 PCA's*

## Imbalance in target values:

We use undersampling and oversampling techniques to address class imbalance in a dataset. Undersampling is a technique used to balance the classes in a dataset by reducing the count of samples in the majority class. This can be performed by randomly selecting samples using more sophisticated techniques such as Cluster Centroids. Oversampling, on the other hand, is a method that involves boosting the number of samples that belong to a dataset's minority class in order to achieve a more equitable distribution of the classes within the dataset. This can be accomplished through the new synthetic samples or through the replication of already existing samples. We use both in model implementation

## Split of dataset:

We are dividing the main dataset into two parts, one contains the feature columns of numerical predictors and a categorical column, which is used for model implementation, other part has participant information which will be used for tableau.

## IV. Data Mining Techniques and Implementation

After that, we apply the following algorithms to the data in order to classify it:

- Logistic Regression
- Decision Tree Classifier
- K-Neighbors Classifier
- Naive Bayes
- Neural network
- Discriminant Analysis

## Flowchart

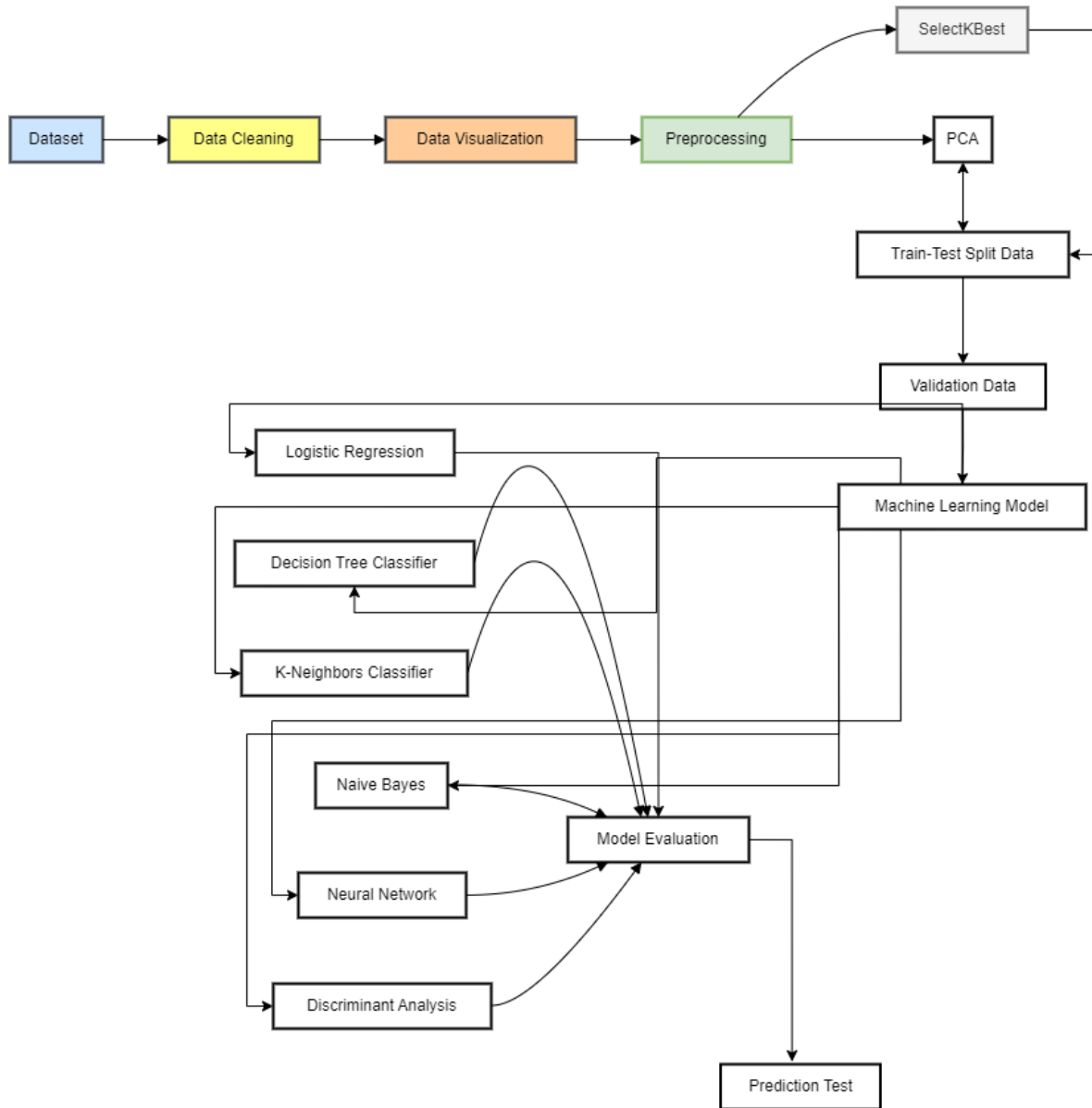


Figure 8: Data Mining Process Flow Chart

## V. Performance Evaluation

On the validation set, we tested each of the algorithms mentioned through their trials by analyzing the ROC Curves, Lift Charts, and Accuracy of the res they achieved on the dataset.

# Performance comparison

Out of all  
"Oversampled Neural  
Network" model  
shown the best  
accuracy

Accuracy score of predict:  
0.9047951563209

Classification report of predict:

	precision	recall	f1-score	support
A	0.94	0.96	0.95	10038
C	0.95	0.91	0.93	35799
E	0.94	0.93	0.93	27942
N	0.90	0.84	0.87	30830
O	0.78	0.93	0.85	10232

micro avg: 0.90 112341  
macro avg: 0.88 0.91 0.89 112341  
weighted avg: 0.91 0.90 0.90 112341

Confusion matrix predict:

```
[[ 9923  32  80  243  80]
 [ 189 50792 351 1584 693]
 [ 389 315 25106 645 587]
 [1151 1338 1096 25888 11537]
 [ 124  49  245  540 54745]]
```

Accuracy score of predict:  
0.696222394470915

Classification report of predict:

	precision	recall	f1-score	support
A	0.82	0.99	0.78	10038
C	0.75	0.59	0.66	35799
E	0.71	0.85	0.87	27942
N	0.78	0.83	0.80	30830
O	0.55	1.00	0.71	10232

micro avg: 0.69 112341  
macro avg: 0.68 0.77 0.70 112341  
weighted avg: 0.71 0.69 0.69 112341

Confusion matrix predict:

```
[[10234  34  34  17]
 [11584 1985 4590 3873 3037]
 [1870 3170 17497 2241 2494]
 [1245 1550 2921 18461 2335]
 [  0  0  0  0 10232]]
```

Classification report of predict:

	precision	recall	f1-score	support
A	0.63	0.95	0.75	10038
C	0.62	0.69	0.75	35799
E	0.62	0.77	0.80	27942
N	0.85	0.77	0.81	30830
O	0.59	0.99	0.71	10232

micro avg: 0.77 112341  
macro avg: 0.74 0.81 0.76 112341  
weighted avg: 0.79 0.77 0.77 112341

Confusion matrix predict:

```
[[ 9581 179 108 185 285]
 [1911 15151 3855 2913 2888]
 [1520 1552 20615 616 1531]
 [1898 1212 1330 15678 1594]
 [ 298  421  289 159 9094]]
```

Accuracy score of predict:  
0.687703164817177

Classification report of predict:

	precision	recall	f1-score	support
A	0.87	0.88	0.87	10038
C	0.89	0.88	0.87	35799
E	0.87	0.83	0.85	27942
N	0.78	0.98	0.88	30830
O	0.80	0.98	0.89	10232

micro avg: 0.86 112341  
macro avg: 0.84 0.87 0.84 112341  
weighted avg: 0.87 0.89 0.89 112341

Confusion matrix predict:

```
[[ 8179 298 828 581 177]
 [2978 13279 5882 2411 1422]
 [2882 1678 17858 1333 1118]
 [1871 5832 5587 17293 814]
 [ 884 1382 1188 879 8088]]
```

Accuracy score of predict:  
0.628881221848830

Classification report of predict:

	precision	recall	f1-score	support
A	0.84	0.79	0.81	10038
C	0.86	0.83	0.84	35799
E	0.88	0.83	0.85	27942
N	0.80	0.93	0.86	30830
O	0.83	0.87	0.85	10232

micro avg: 0.83 112341  
macro avg: 0.81 0.84 0.83 112341  
weighted avg: 0.86 0.82 0.83 112341

Confusion matrix predict:

```
[[ 7858 278 281 789 870]
 [4889 18111 9380 4821 4282]
 [5882 1219 13872 2789 1288]
 [9462 1807 2893 14222 2876]
 [1817 837 1183 889 8898]]
```

Accuracy score of predict:  
0.6999910008989127

Classification report of predict:

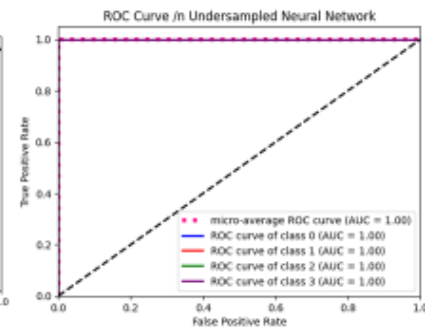
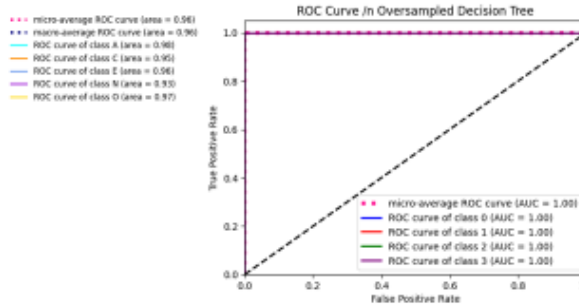
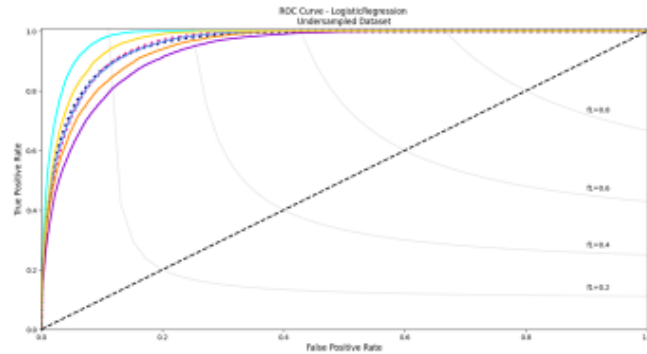
	precision	recall	f1-score	support
A	1.00	1.00	1.00	10038
C	1.00	1.00	1.00	35799
E	1.00	1.00	1.00	27942
N	1.00	1.00	1.00	30830
O	1.00	1.00	1.00	10232

micro avg: 1.00 112341  
macro avg: 1.00 1.00 1.00 112341  
weighted avg: 1.00 1.00 1.00 112341

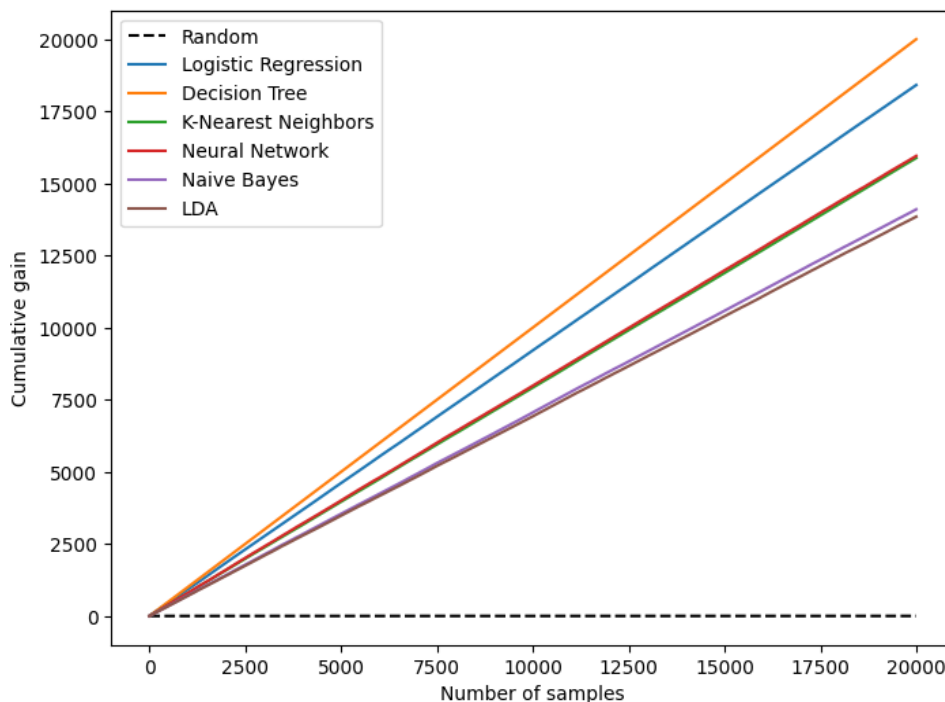
Confusion matrix predict:

```
[[10038  0  0  0  0]
 [ 0 35799  0  0  0]
 [ 0  0 27942  0  0]
 [ 0  0  0 30830  0]
 [ 0  0  0  0 10232]]
```

Accuracy score of predict:  
0.7706283278632778



## Gain Charts



We then created a performance table to compare them all, the Oversampled Neural network showed best res, and we use it for further prediction of new dataset.

## VI. Discussion and Recommendation

Classification report of Oversampled Neural network data,

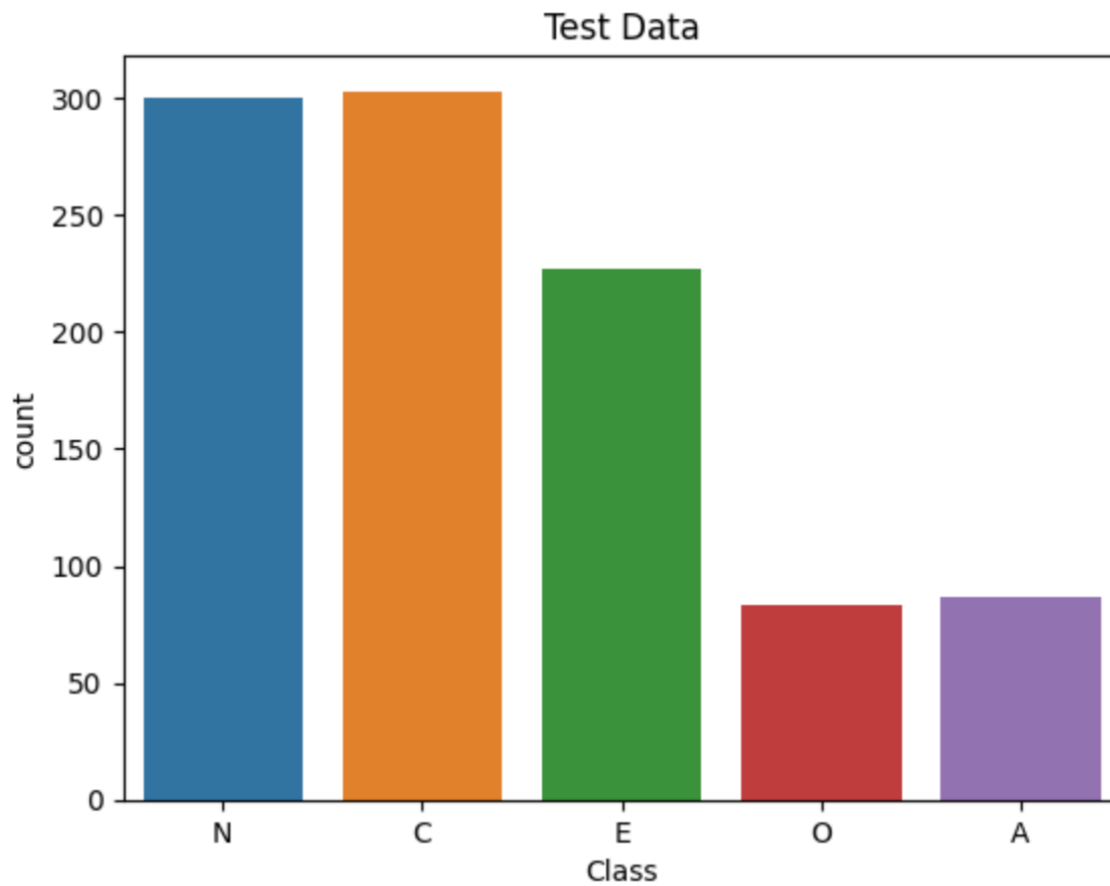
	precision	recall	f1-score	support
A	1.00	1.00	1.00	10338
C	1.00	1.00	1.00	33799
E	1.00	1.00	1.00	27042
N	1.00	1.00	1.00	30830
O	1.00	1.00	1.00	10232
accuracy			1.00	112241
macro avg	1.00	1.00	1.00	112241
weighted avg	1.00	1.00	1.00	112241

Confusion matrix

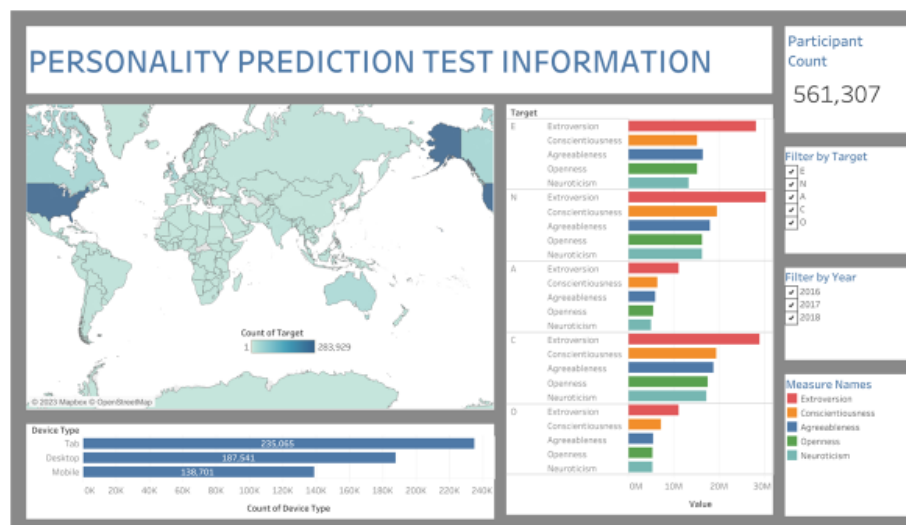
```
[[10338  0  0  0  0]
 [  0 33799  0  0  0]
 [  0  0 27042  0  0]
 [  0  0  0 30830  0]
 [  0  0  0  0 10232]]
```

## VII. Summary

In this analysis, we put a wide variety of machine learning models through their paces to see how well they can handle a multi classification problem. We train the model on a dataset that has been standardized. Used the Oversampled Neural network model and got target values.



## Tableau Desktop



## Appendix: Python Code for use case study

### # Case Study

```
# -*- coding: utf-8 -*-
```

```
"""DM_Project_Personality_Prediction_final (1).ipynb
```

Automatically generated by Colaboratory.

Original file is located at

```
https://colab.research.google.com/drive/1MDMm4Cr-b-lBloqzLWVSnR0ExDhz0TsB
"""
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
pip install geopandas
```

```
pip install pycountry
```

```
"""
```

```
import pandas as pandd
import numpy as nanp
import pycountry
import warnings
import matplotlib.pyplot as plott
import seaborn as sens
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
from sklearn.model_selection import train_test_split
from pandas.plotting import scatter_matrix, parallel_coordinates
from sklearn.feature_selection import SelectKBest, chi2, f_regression
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
warnings.filterwarnings("ignore")
```

```
Fivefac1 = pandd.read_csv("/content/drive/MyDrive/BigFivePersonalitydata.csv")
Fivefac1.head()
```

```
Fivefac = Fivefac1
```

```
#Datatype of each column
dict(Fivefac.dtypes)
```

```
last_10_cols = Fivefac.iloc[:, -12:]
data_types = last_10_cols.dtypes
```

```
print(data_types)
```

```
Fivefac["date_column"] = pandd.to_datetime(Fivefac["dateload"], format="%Y-%m-%d %H:%M:%S")
Fivefac = Fivefac.drop('dateload', axis=1)
```



```

Fivefac["lat_appx_lots_of_err"] = pandd.to_numeric(Fivefac["lat_appx_lots_of_err"],
errors="coerce").astype(float)
Fivefac["long_appx_lots_of_err"] = pandd.to_numeric(Fivefac["long_appx_lots_of_err"],
errors="coerce").astype(float)
Fivefac["country"] = Fivefac["country"].astype(str)
Fivefac["target"] = Fivefac["target"].astype(str)

#Data type information for each variable in our dataset
Fivefac.info()

Fivefac.describe(include='all')

pandd.DataFrame({'mean':Fivefac.mean(), 'sd':Fivefac.std(), 'min':Fivefac.min(),
                'max':Fivefac.max(), 'median':Fivefac.median(), 'length':len(Fivefac),
                'miss.val':Fivefac.isnull().sum()})

Fivefac = Fivefac.iloc[:, 2:]
Fivefac.head()

# Create a dictionary of country code to country name mappings
country_dict = {}
for country in pycountry.countries:
    country_dict[country.alpha_2] = country.name

# Map the Alpha-2 codes to country names
Fivefac['Countries'] = Fivefac['country'].map(country_dict)

#Remove Country column
Fivefac = Fivefac.drop("country", axis=1)

Fivefac["screen_size"] = (Fivefac["screenw"] ** 2 + Fivefac["screenh"] ** 2) ** 0.5

#Removing Screen width and screen height column
Fivefac = Fivefac.drop("screenw", axis=1)
Fivefac = Fivefac.drop("screenh", axis=1)

tab_threshold = 7
phone_min_threshold = 4
phone_max_threshold = 7

# Create a new column for device type based on screen size
Fivefac["device_type"] = pandd.cut(Fivefac["screen_size"],
                                bins=[0, phone_min_threshold, phone_max_threshold, float("inf")],
                                labels=["Desktop", "Mobile", "Tab"])

Fivefac["device_type"][Fivefac.screen_size <=800] = "Mobile"
Fivefac["device_type"][(Fivefac.screen_size <= 1600) & (Fivefac.screen_size>800)] = "Tab"
Fivefac["device_type"][Fivefac.screen_size >1600] = "Desktop"

fig = plott.figure(figsize = (10, 5))

# creating the bar plot
counts = Fivefac["device_type"].value_counts()

```

```

# Create a bar plot
plott.bar(counts.index, counts.values)

# Add labels and title
plott.xlabel('Device Type')
plott.ylabel('Count')
plott.title('Distribution of Devices')

# Show the plot
plott.show()

counts = Fivefac["IPC"].value_counts()

# Create a bar plot
plott.bar(counts.index, counts.values)

# Add labels and title
plott.xlabel('IP Count')
plott.ylabel('Count')
plott.title('Distribution of IP address count')

# Show the plot
plott.show()

Fivefac.drop(Fivefac.index[Fivefac['IPC'] != 1], inplace = True)
Fivefac['IPC'].describe()
Fivefac = Fivefac.drop("IPC", axis=1)

Fivefac["Time consumed"] = Fivefac["introelapse"] + Fivefac["testelapse"] + Fivefac["endelapse"]
Fivefac = Fivefac.drop("introelapse", axis=1)
Fivefac = Fivefac.drop("testelapse", axis=1)
Fivefac = Fivefac.drop("endelapse", axis=1)

Fivefac = Fivefac.drop_duplicates()

nullv = Fivefac.isnull().sum() + Fivefac.isna().sum() #Null and nun Values
null_percent = nullv* 100 / len(Fivefac) #Percentage of null values
nullv_df = pandd.DataFrame({'column_name': Fivefac.columns, 'nullv_values': nullv, 'percent_nullv (%)':
null_percent})
nullv_df.sort_values('nullv_values', inplace=True, ascending=False)
nullv_df

Fivefac = Fivefac.dropna(subset=['lat_appx_lots_of_err'])
Fivefac = Fivefac.dropna(subset=['long_appx_lots_of_err'])

import requests

# select rows with missing 'country' values
missing_data = Fivefac[Fivefac['Countries'].isna()]

# loop through each row and get the country value using the 'lat' and 'long' columns
for index, row in missing_data.iterrows():
    if (str(row["lat_appx_lots_of_err"]) != "nan") & (str(row["long_appx_lots_of_err"]) != "nan"):
        url =

```

```

f'https://nominatim.openstreetmap.org/reverse?format=jsonv2&lat={row["lat_appx_lots_of_err"]}&lon={row["lon_appx_lots_of_err"]}'
try:
    response = requests.get(url).json()
except:
    pass
country = response['address']['country']
Fivefac.at[index, 'Countries'] = country

Fivefac = Fivefac.drop("lat_appx_lots_of_err", axis=1)
Fivefac = Fivefac.drop("long_appx_lots_of_err", axis=1)

#Dropping the Null values
Fivefac = Fivefac.dropna()
Fivefac.isnull().sum().any()

Fivefac.iloc[:,50:100] = Fivefac.iloc[:,50:100][(Fivefac.iloc[:,50:100] > 0).all(1)] #Answers should be between 1-5.
Fivefac = Fivefac.dropna()

#The time columns are measured in milliseconds, and needs to convert them to seconds format.
Fivefac.iloc[:,50:100] = Fivefac.iloc[:,50:100].divide(1000, axis = 'rows')
Fivefac.iloc[:,50:100]

Fivefac.describe(include='all')

bestfeatures = SelectKBest(k=60)
fit = bestfeatures.fit(Fivefac.iloc[:,0:100],Fivefac['target'])
dfscores = pandd.DataFrame(fit.scores_)
dfcolumns = pandd.DataFrame(Fivefac.drop('target',axis=1).columns)

featureScores = pandd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Feature','Score']
(featureScores.sort_values(by='Score',ascending=False)).T

categories = {'Extroversion': 'EXT', 'Neuroticism': 'EST', 'Openness': 'OPN', 'Conscientiousness': 'CSN',
'Agreeableness': 'AGR'}
# Creating a dictionary for the category names and their corresponding prefixes.

for category, prefix in categories.items():
    Fivefac[category] = Fivefac.filter(regex=f'^{prefix}', axis=1).sum(axis=1)
# Using the filter method to select the columns with the given prefix, then summing them along the row axis.

#Columns having data of participants response on scale of 1 to 5
question_Fivefac = (Fivefac.iloc[:, 0:50])
question_Fivefac = question_Fivefac[(question_Fivefac > 0).all(1)] #Answers should be between 1-5.
question_Fivefac['target'] = Fivefac['target']
question_Fivefac

catpoint_types = ['EXT', 'EST', 'AGR', 'CSN', 'OPN']
targets = question_Fivefac['target'].unique()

for q in catpoint_types:
    questions = [f"{q}{i}" for i in range(1, 11)]

```

```

fig, axis = plott.subplots(ncols=2, nrows=5, figsize=(20, 12))
fig.suptitle(q)
for i, question in enumerate(questions):
    row_idx = i % 5
    col_idx = i // 5
    ax = axis[row_idx, col_idx]
    ax.set_title(question)
    targets_data = [question_Fivefac.loc[question_Fivefac['target'] == target,
question].value_counts().sort_index() for target in targets]
    cumsum = nanp.cumsum(targets_data, axis=0)
    for j in range(len(targets_data)):
        if j == 0:
            ax.bar(targets_data[j].index, targets_data[j], label=targets[j])
        else:
            ax.bar(targets_data[j].index, targets_data[j], bottom=cumsum[j - 1], label=targets[j])
    ax.legend()

stats_df= pandd.DataFrame({'mean':question_Fivefac.mean(),
'sd':question_Fivefac.std(),'min':question_Fivefac.min(),
    'max':question_Fivefac.max(), 'median':question_Fivefac.median(), 'length':len(question_Fivefac),
    'Missing_values':question_Fivefac.isnull().sum()})
stats_df.T

#Dropping columns having participants responses
Fivefac.drop(Fivefac.columns[0:50], axis=1, inplace=True)
Fivefac.head()

corr = Fivefac.drop('target',axis=1).corr(method='pearson')
fig,ax = plott.subplots(1,1,figsize=(20,20))
sens.heatmap(corr,cmap='Blues')
plott.title('Correlation Matrix')
plott.show()

#Heatmap of main factors
hmapanddf = Fivefac[['Extroversion', 'Neuroticism', 'Openness', 'Conscientiousness', 'Agreeableness']]
corrhe= hmapanddf.corr().round(2)
plottt.figure(figsize=(10,8))
sens.heatmap(corrhe,annot=True,center=0)

axees = scatter_matrix(hmapanddf, alpha = 0.6, figsize = (8,8), diagonal = 'kde')
try:
    corr = hmapanddf.corr().values()

    for ii, ij in zip(*plott.nanp.triu_indices_from(axees, k=1)):
        axees[ii,ij].annotate('%0.3f' %corr[ii,ij], (0.8,0.9),
            xycrds = "axes fraction", ha = 'center', va='center')
except Exception:
    pass
plott.show()

Fivefac.iloc[:,50:].to_csv('Participants_Fivefac.csv', index=False)

Fivefac.iloc[:,50:]

```

```

question_Fivefac = pandd.read_csv("question_Fivefac.csv")

question_Fivefac.shape

#question_Fivefac['target'] = Fivefac['target']
# Define the target variable and independent variables
target_variable = 'target'
independent_variables = ['EXT1', 'EXT2', 'EXT3', 'EXT4', 'EXT5', 'EXT6', 'EXT7',
'EXT8', 'EXT9', 'EXT10', 'EST1', 'EST2', 'EST3', 'EST4', 'EST5', 'EST6', 'EST7', 'EST8',
    'EST9', 'EST10', 'AGR1', 'AGR2', 'AGR3', 'AGR4', 'AGR5', 'AGR6', 'AGR7',
    'AGR8', 'AGR9', 'AGR10', 'CSN1', 'CSN2', 'CSN3', 'CSN4', 'CSN5', 'CSN6',
    'CSN7', 'CSN8', 'CSN9', 'CSN10', 'OPN1', 'OPN2', 'OPN3', 'OPN4', 'OPN5',
    'OPN6', 'OPN7', 'OPN8', 'OPN9', 'OPN10']

# Compute the correlation coefficients between the target variable and each independent variable
correlation_matrix = question_Fivefac[independent_variables + [target_variable]].corr()

# Display the significant correlations
print('Correlation Matrix:')
correlation_matrix

sens.heatmap(correlation_matrix,xticklabels = correlation_matrix.columns, yticklabels =
correlation_matrix.columns,cmap='Blues', linewidth=.5)
plott.title('Correlation Matrix')
plott.show()

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
predictors_scaled = scaler.fit_transform(question_Fivefac.drop(['target'], axis=1))

# Perform PCA to reduce the dimensionality of the predictors
pca = PCA()
pca.fit(predictors_scaled)
explained_variance = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance)
num_components = np.argmax(cumulative_variance >= 0.95) + 1
print(f"Number of components required for 95% of the variance: {num_components}")

# Project the data onto the top k principal components
pca = PCA(n_components=num_components)
predictors_pca = pca.fit_transform(predictors_scaled)

predictors = question_Fivefac.drop(['target'], axis=1)
feature_names = predictors.columns

# Get the top 43 principal components
top_components = pca.components_[0:43, :]

top_variable_indices = [list(abs(top_components[i]).argsort()[::-1][0:43]) for i in range(43)]

top_variable_names = [[feature_names[index] for index in indices] for indices in top_variable_indices]

# Print the names of the top 43 variables for each principal component

```

```

for i, component in enumerate(top_variable_names):
    print(f"Top 43 variables for Principal Component {i+1}: {' '.join(component)}")

Sumarizepc = pandas.DataFrame({'Standard Deviation': np.sqrt(pca.explained_variance_),
                              'Proportion of variance':pca.explained_variance_ratio_,
                              'Cumulative proportion': np.cumsum(pca.explained_variance_ratio_)})
Sumarizepc = Sumarizepc.transpose()
Sumarizepc.columns = [f'PC{i}' for i in range(1, 44)]
Sumarizepc.round(3)

listpca=top_variable_names[42]

questionanpcs_Fivefac = question_Fivefac[listpca]

questionanpcs_Fivefac['target'] = question_Fivefac['target']
questionanpcs_Fivefac

under_traimo = {}
Over_traimo = {}

models_under = {}
models_Over ={}

under_traimopca = {}
Over_traimopca = {}

models_underpca = {}
models_Overpca ={}

X, y = question_Fivefac.drop('target',axis=1), question_Fivefac['target']
X_pca, y_pca = questionanpcs_Fivefac.drop('target',axis=1), questionanpcs_Fivefac['target']
# Perform random undersampling
rus = RandomUnderSampler()
X_undersampled, y_undersampled = rus.fit_resample(X, y)
X_undersampled_pca, y_undersampled_pca = rus.fit_resample(X_pca, y_pca)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=0, shuffle=True, stratify = y)
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y_pca, test_size=0.2,
                                                                    random_state=0, shuffle=True, stratify = y_pca)

"""Once we have performed undersampling or oversampling, we can proceed to implement the following
models:

**1.Logistic Regression**
"""

from sklearn.linear_model import LogisticRegression

# Create the model to make predictions
lr_model = LogisticRegression(solver='lbfgs', max_iter=150)

lr_model.fit(X_undersampled, y_undersampled)
y_pred_lr = lr_model.predict(X_test)

```

```

under_traimo['UnderLogistic'] = lr_model
models_under['predlr'] = y_pred_lr

lr_modelpca = LogisticRegression(solver='lbfgs', max_iter=150)
lr_modelpca.fit(X_undersampled_pca, y_undersampled_pca)
y_pred_lr_pca = lr_modelpca.predict(X_test_pca)

under_traimopca['UnderLogistic'] = lr_modelpca
models_underpca['predlr'] = y_pred_lr_pca

from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Create the models
dtc_model = DecisionTreeClassifier()
knc_model = KNeighborsClassifier()
nb_model = GaussianNB()
nn_model = MLPClassifier()
lda_model = LinearDiscriminantAnalysis()

# Train the models on the undersampled data
dtc_model.fit(X_undersampled, y_undersampled)
knc_model.fit(X_undersampled, y_undersampled)
nb_model.fit(X_undersampled, y_undersampled)
nn_model.fit(X_undersampled, y_undersampled)
lda_model.fit(X_undersampled, y_undersampled)

# Make predictions on the test data
y_pred_dtc = dtc_model.predict(X_test)
y_pred_knc = knc_model.predict(X_test)
y_pred_nb = nb_model.predict(X_test)
y_pred_nn = nn_model.predict(X_test)
y_pred_lda = lda_model.predict(X_test)

# Store the models and predictions in dictionaries
under_traimo['UnderDecisionTree'] = dtc_model
models_under['pred_dtc'] = y_pred_dtc

under_traimo['UnderKNeighbors'] = knc_model
models_under['pred_knc'] = y_pred_knc

under_traimo['UnderNaiveBayes'] = nb_model
models_under['pred_nb'] = y_pred_nb

under_traimo['UnderNeuralNetwork'] = nn_model
models_under['pred_nn'] = y_pred_nn

under_traimo['UnderLDA'] = lda_model
models_under['pred_lda'] = y_pred_lda

```

```

# Repeat for the PCA dataset
dtc_modelpca = DecisionTreeClassifier()
knc_modelpca = KNeighborsClassifier()
nb_modelpca = GaussianNB()
nn_modelpca = MLPClassifier()
lda_modelpca = LinearDiscriminantAnalysis()

dtc_modelpca.fit(X_undersampledpca, y_undersampledpca)
knc_modelpca.fit(X_undersampledpca, y_undersampledpca)
nb_modelpca.fit(X_undersampledpca, y_undersampledpca)
nn_modelpca.fit(X_undersampledpca, y_undersampledpca)
lda_modelpca.fit(X_undersampledpca, y_undersampledpca)

y_pred_dtcpca = dtc_modelpca.predict(X_testpca)
y_pred_kncpca = knc_modelpca.predict(X_testpca)
y_pred_nbpca = nb_modelpca.predict(X_testpca)
y_pred_nnanpca = nn_modelpca.predict(X_testpca)
y_predLDAPCA = lda_modelpca.predict(X_testpca)

under_traimopca['UnderDecisionTree'] = dtc_modelpca
models_underpca['preddtcpca'] = y_pred_dtcpca

under_traimopca['UnderKNeighbors'] = knc_modelpca
models_underpca['predkncpca'] = y_pred_kncpca

under_traimopca['UnderNaiveBayes'] = nb_modelpca
models_underpca['prednbpca'] = y_pred_nbpca

under_traimopca['UnderNeuralNetwork'] = nn_modelpca
models_underpca['prednnanpca'] = y_pred_nnanpca

under_traimo['UnderLDAPca'] = lda_modelpca
models_under['predlda'] = y_predLDAPCA

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
def errorsis(y_pred,modelname):
    # Calculate accuracy
    print("Accuracy score of "+ modelname +"\n")
    print(accuracy_score(y_test, y_pred))

    # Calculate precision, recall, and F1 score
    print("Classification report of "+ modelname +"\n")
    print(classification_report(y_test, y_pred))

    # Calculate the confusion matrix
    print("Confusion matrix "+ modelname +"\n")
    print(confusion_matrix(y_test, y_pred))

def pr_roc_curves_multiclass(estmodel,X,y,label = None):
    y = label_binarize(y, classes=list(estmodel.classes_))
    noclas = y.shape[1]

    # shuffle and split training and test sets

```



```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2, random_state=0,
                                                    shuffle=True, stratify = y)

y_score = estmodel.decision_function(X_test)

estfpr = dict()
esttpr = dict()
estroc = dict()

for i in range(noclas):
    estfpr[i], esttpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    estroc[i] = auc(estfpr[i], esttpr[i])

# Compute micro-average ROC curve and ROC area and aggregate all false positive rates
estfpr["micro"], esttpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
estroc["micro"] = auc(estfpr["micro"], esttpr["micro"])
all_estfpr = nanp.unique(nanp.concatenate([estfpr[i] for i in range(noclas)]))

mean_esttpr = nanp.zeros_like(all_estfpr)
for i in range(noclas):
    mean_esttpr += interp(all_estfpr, estfpr[i], esttpr[i])

# Average it and compute AUC
mean_esttpr /= noclas
estfpr["macro"] = all_estfpr
esttpr["macro"] = mean_esttpr
estroc["macro"] = auc(estfpr["macro"], esttpr["macro"])

fig, ax = plt.subplots(figsize=(16,8))
colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'darkviolet', 'gold', 'blue'])
lw = 2

# Plot Precision - Recall curves
# Plot iso-F1 curves
fscs = nanp.linspace(0.2, 0.8, num=4)
lines = []
labels = []
for fsc in fscs:
    x = nanp.linspace(0.01, 1)
    y = fsc * x / (2 * x - fsc)
    l, = ax.plot(x[y >= 0], y[y >= 0], color='gray', alpha=0.2)
    ax.annotate('f1={0:0.1f}'.format(fsc), xy=(0.9, y[45] + 0.02))

lines.append(l)
labels.append('iso-f1 curves')

# Plot ROC Curves
ax.plot(estfpr["micro"], esttpr["micro"],
        color='deeppink', linestyle=':', linewidth=4,
        label='micro-average ROC curve (area = {0:0.2f})'.format(estroc["micro"]))

ax.plot(estfpr["macro"], esttpr["macro"],
        color='navy', linestyle=':', linewidth=4,

```

```

label='macro-average ROC curve (area = {0:0.2f})'.format(estroc["macro"]))

for i, color in zip(range(noclas), colors):
    class_ref = list(estmodel.classes_)[i]

    # Same for ROC curves
    ax.plot(estfpr[i], esttpr[i], color=color, lw=lw,
            label='ROC curve of class {0} (area = {1:0.2f})'.format(class_ref,estroc[i]))

ax.plot([0, 1], [0, 1], 'k--', lw=lw)
ax.set_xlim([0.0, 1.0])
ax.set_ylim([0.0, 1.01])
ax.set_xlabel('False Positive Rate', fontsize=12)
ax.set_ylabel('True Positive Rate', fontsize=12)
ax.set_title('ROC Curve - {type(estmodel).__name__} {label}', fontsize=12)
ax.legend(loc=(0, -.45), prop=dict(size=12), frameon=False, fontsize=12)

plott.show()

"""**Oversampling**"""

X, y = question_Fivefac.drop('target', axis=1), question_Fivefac['target']
X_pca, y_pca = questionanpcs_Fivefac.drop('target', axis=1), questionanpcs_Fivefac['target']
# Perform random Oversampling
rus = RandomOverSampler()
X_Oversampled, y_Oversampled = rus.fit_resample(X, y)
X_Oversampled_pca, y_Oversampled_pca = rus.fit_resample(X_pca, y_pca)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=0, shuffle=True, stratify=y)
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y_pca, test_size=0.2,
                                                                    random_state=0, shuffle=True, stratify=y_pca)

lr_model = LogisticRegression(solver='lbfgs', max_iter=150)

lr_model.fit(X_Oversampled, y_Oversampled)
y_pred_lr = lr_model.predict(X_test)

Over_traimo['OverLogistic'] = lr_model
models_Over['predlr'] = y_pred_lr

lr_model_pca = LogisticRegression(solver='lbfgs', max_iter=150)
lr_model_pca.fit(X_Oversampled_pca, y_Oversampled_pca)
y_pred_lr_pca = lr_model_pca.predict(X_test_pca)

Over_traimopca['OverLogistic'] = lr_model_pca
models_Overpca['predlr'] = y_pred_lr_pca

# Create the models
dtc_model = DecisionTreeClassifier()
knc_model = KNeighborsClassifier()

```

```
nb_model = GaussianNB()
nn_model = MLPClassifier()
lda_model = LinearDiscriminantAnalysis()
```

```
# Train the models on the Oversampled data
dtc_model.fit(X_Oversampled, y_Oversampled)
knc_model.fit(X_Oversampled, y_Oversampled)
nb_model.fit(X_Oversampled, y_Oversampled)
nn_model.fit(X_Oversampled, y_Oversampled)
lda_model.fit(X_Oversampled, y_Oversampled)
```

```
# Make predictions on the test data
y_pred_dtc = dtc_model.predict(X_test)
y_pred_knc = knc_model.predict(X_test)
y_pred_nb = nb_model.predict(X_test)
y_pred_nn = nn_model.predict(X_test)
y_pred_lda = lda_model.predict(X_test)
```

```
# Store the models and predictions in dictionaries
Over_traimo['OverDecisionTree'] = dtc_model
models_Over['preddtc'] = y_pred_dtc
```

```
Over_traimo['OverKNeighbors'] = knc_model
models_Over['predknc'] = y_pred_knc
```

```
Over_traimo['OverNaiveBayes'] = nb_model
models_Over['prednb'] = y_pred_nb
```

```
Over_traimo['OverNeuralNetwork'] = nn_model
models_Over['prednn'] = y_pred_nn
```

```
Over_traimo['OverLDA'] = lda_model
models_Over['predlda'] = y_pred_lda
```

```
# Repeat for the PCA dataset
dtc_modelpca = DecisionTreeClassifier()
knc_modelpca = KNeighborsClassifier()
nb_modelpca = GaussianNB()
nn_modelpca = MLPClassifier()
lda_modelpca = LinearDiscriminantAnalysis()
```

```
dtc_modelpca.fit(X_Oversampledpca, y_Oversampledpca)
knc_modelpca.fit(X_Oversampledpca, y_Oversampledpca)
nb_modelpca.fit(X_Oversampledpca, y_Oversampledpca)
nn_modelpca.fit(X_Oversampledpca, y_Oversampledpca)
lda_modelpca.fit(X_Oversampledpca, y_Oversampledpca)
```

```
y_pred_dtcpca = dtc_modelpca.predict(X_testpca)
y_pred_kncpca = knc_modelpca.predict(X_testpca)
y_pred_nbpca = nb_modelpca.predict(X_testpca)
y_pred_nnapca = nn_modelpca.predict(X_testpca)
y_predLDAPCA = lda_modelpca.predict(X_testpca)
```

```
Over_traimopca['OverDecisionTree'] = dtc_modelpca
models_Overpca['preddtcpca'] = y_pred_dtcpca
```

```
Over_traimopca['OverKNeighbors'] = knc_modelpca
models_Overpca['predkncpca'] = y_pred_kncpca
```

```
Over_traimopca['OverNaiveBayes'] = nb_modelpca
models_Overpca['prednbpca'] = y_pred_nbpca
```

```
Over_traimopca['OverNeuralNetwork'] = nn_modelpca
models_Overpca['prednnanpca'] = y_pred_nnanpca
```

```
Over_traimo['OverLDAPca'] = lda_modelpca
models_Over['predlda'] = y_predLDAPCA
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
for i, j in models_under.items():
    errorsis(j,i)
```

```
for i, j in models_underpca.items():
    errorsis(j,i)
```

```
for i, j in models_Over.items():
    errorsis(j,i)
```

```
for i, j in models_Overpca.items():
    errorsis(j,i)
```

```
from itertools import cycle
pr_roc_curves_multiclass(under_traimo['UnderLogistic'],X_undersampled,y_undersampled, label =
'\nUndersampled Dataset')
```

```
under_traimo
```

```
def plot_lift_curve(clf, X_test, y_test, name='Classifier'):
    # Predict probabilities for test set
    y_prob = clf.predict_proba(X_test)

    # Get precision, recall, and thresholds
    precision, recall, thresholds = precision_recall_curve(y_test.ravel(), y_prob[:, 1].ravel())

    # Calculate ratios for lift curve
    lift_ratio = (recall / nanp.mean(y_test))

    # Calculate area under curve (AUC)
    auc_score = auc(lift_ratio, precision)

    # Plot lift curve
    plott.plot(lift_ratio, precision, label='{ } (AUC = {:.3f})'.format(name, auc_score))
    plott.xlabel('Lift Ratio')
    plott.ylabel('Precision')
    plott.title('Lift Curve')
    plott.legend(loc='best')
```

```
plott.show()
```

```
# label_encoder object knows
# how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
# Get the true labels for the test set
y_true=label_encoder.fit_transform(y_true)
y_true = y_test.values
# Get the predicted probabilities for each model
y_pred_probs_lr = lr_model.predict_proba(X_test)
y_pred_probs_dtc = dtc_model.predict_proba(X_test)
y_pred_probs_knc = knc_model.predict_proba(X_test)
y_pred_probs_nb = nb_model.predict_proba(X_test)
y_pred_probs_nn = nn_model.predict_proba(X_test)
y_pred_probs_lda = lda_model.predict_proba(X_test)

# Combine the predicted probabilities into one array
y_pred_probs = np.stack((y_pred_probs_lr, y_pred_probs_dtc, y_pred_probs_knc, y_pred_probs_nb,
y_pred_probs_nn, y_pred_probs_lda), axis=-1)

# Sort the predicted probabilities for each observation in descending order
sorted_probs = np.sort(y_pred_probs, axis=0)[::-1]

# Get the true labels for the test set
y_true=label_encoder.fit_transform(y_true)
y_true = y_test.values

# Calculate the cumulative number of observations and the cumulative number of correct predictions for each
class
cumulative_obs = np.arange(1, len(y_true) + 1)
cumulative_correct = np.zeros((len(y_true), 6))

for i in range(6):
    for j in range(len(y_true)):
        if np.argmax(sorted_probs[j,:,i]) == y_true[j]:
            cumulative_correct[j,:,i] += 1

# Plot the gain chart
plott.figure(figsize=(8, 6))
plott.plot(cumulative_obs/(cumulative_obs[-1]*10), cumulative_correct[:,0]/cumulative_correct[-1,0],
label='Logistic Regression')
plott.plot(cumulative_obs/(cumulative_obs[-1]*10), cumulative_correct[:,1]/cumulative_correct[-1,1],
label='Decision Tree')
plott.plot(cumulative_obs/(cumulative_obs[-1]*10), cumulative_correct[:,2]/cumulative_correct[-1,2], label='K
Neighbors')
plott.plot(cumulative_obs/(cumulative_obs[-1]*10), cumulative_correct[:,3]/cumulative_correct[-1,3],
label='Naive Bayes')
plott.plot(cumulative_obs/(cumulative_obs[-1]*10), cumulative_correct[:,4]/cumulative_correct[-1,4],
label='Neural Network')
plott.plot(cumulative_obs/(cumulative_obs[-1]*10), cumulative_correct[:,5]/cumulative_correct[-1,5],
label='LDA')
plott.plot([0, 1], [0, 1], 'k--', label='Random')
plott.xlabel('Cumulative Fraction of Observations')
```

```

plott.ylabel('Cumulative Fraction of Correct Predictions')
plott.title('Gain Chart - Oversampled Data')
plott.legend()
plott.show()

# Define a function to create the gain chart
def gain_chart(y_true, y_proba, title):

    # Create a sorted array of the true labels and corresponding predicted probabilities
    sort_idx = np.argsort(y_proba)[::-1]
    y_true_sorted = y_true[sort_idx]
    y_proba_sorted = y_proba[sort_idx]

    # Compute the cumulative sums of the true labels and predicted probabilities
    cumsum_true = np.cumsum(y_true_sorted)
    cumsum_proba = np.cumsum(y_proba_sorted)

    # Compute the total number of positive labels and the total number of samples
    n_pos = np.sum(y_true)
    n_samples = len(y_true)

    # Compute the random gain line and the perfect gain line
    random_gain = np.linspace(0, n_pos, n_samples)
    perfect_gain = cumsum_true / n_pos

    # Compute the actual gain line
    actual_gain = cumsum_true / n_pos - cumsum_proba / n_samples

    # Plot the gain chart
    plott.plot(np.linspace(0, 1, n_samples), random_gain, label='Random')
    plott.xlabel('Number of samples')
    plott.ylabel('Cumulative gain')
    plott.title(title)
    plott.legend()
    plott.show()
names=['Logistic Regression',
       'Decision Tree',
       'K-Nearest Neighbors',
       'Neural Network',
       'Naive Bayes',
       'LDA']
# Call the gain_chart function for each model
models = {'OverLogistic': models_Over['predlr'],
          'OverDecisionTree': models_Over['preddtc'],
          'OverKNeighbors': models_Over['predknc'],
          'OverNaiveBayes': models_Over['prednb'],
          'OverNeuralNetwork': models_Over['prednn'],
          'OverLDA': models_Over['predlda']}
for model_name, y_pred in models.items():
    y_proba = Over_traimo[model_name].predict_proba(X_test)[: , 1]
    gain_chart(y_test, y_proba, title=names)

from sklearn.datasets import make_classification
from sklearn.neural_network import MLPClassifier

```

```

from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from scipy import interp

# binarize the labels
y_bin = label_binarize(y_undersampled, classes=np.unique(y_undersampled))
# predict probabilities for the test data
y_pred_prob = under_traimo['UnderNeuralNetwork'].predict_proba(X_undersampled)

estfpr = dict()
esttpr = dict()
estroc = dict()
for i in range(under_traimo['UnderNeuralNetwork'].classes_.size):
    estfpr[i], esttpr[i], _ = roc_curve(y_bin[:, i], y_pred_prob[:, i])
    estroc[i] = auc(estfpr[i], esttpr[i])

# calculate the micro-average ROC curve and AUC score
estfpr["micro"], esttpr["micro"], _ = roc_curve(y_bin.ravel(), y_pred_prob.ravel())
estroc["micro"] = auc(estfpr["micro"], esttpr["micro"])

# plot the ROC curves for each class and the micro-average ROC curve
plott.figure()
lw = 2
plott.plot(estfpr["micro"], esttpr["micro"], color='deeppink', linestyle=':', linewidth=4, label='micro-average ROC
curve (AUC = {0:0.2f})'.format(estroc["micro"]))
colors = ['blue', 'red', 'green', 'purple']
for i, color in zip(range(under_traimo['UnderNeuralNetwork'].classes_.size), colors):
    plott.plot(estfpr[i], esttpr[i], color=color, lw=lw, label='ROC curve of class {0} (AUC = {1:0.2f})'.format(i,
estroc[i]))

# plot the baseline model
plott.plot([0, 1], [0, 1], 'k--', lw=lw)
plott.xlim([0.0, 1.0])
plott.ylim([0.0, 1.05])
plott.xlabel('False Positive Rate')
plott.ylabel('True Positive Rate')
plott.title('ROC Curve /n Undersampled Neural Network')
plott.legend(loc="lower right")
plott.show()

# binarize the labels
y_bin = label_binarize(y_OversampledPCA, classes=np.unique(y_OversampledPCA))
# predict probabilities for the test data
y_pred_prob = Over_traimopca['OverDecisionTree'].predict_proba(X_OversampledPCA)

estfpr = dict()
esttpr = dict()
estroc = dict()
for i in range(Over_traimopca['OverDecisionTree'].classes_.size):
    estfpr[i], esttpr[i], _ = roc_curve(y_bin[:, i], y_pred_prob[:, i])

```

```

estroc[i] = auc(estfpr[i], esttpr[i])

# calculate the micro-average ROC curve and AUC score
estfpr["micro"], esttpr["micro"], _ = roc_curve(y_bin.ravel(), y_pred_prob.ravel())
estroc["micro"] = auc(estfpr["micro"], esttpr["micro"])

# plot the ROC curves for each class and the micro-average ROC curve
plott.figure()
lw = 2
plott.plot(estfpr["micro"], esttpr["micro"], color='deeppink', linestyle=':', linewidth=4, label='micro-average ROC
curve (AUC = {0:0.2f})'.format(estroc["micro"]))
colors = ['blue', 'red', 'green', 'purple']
for i, color in zip(range(Over_trainopca['OverDecisionTree'].classes_.size), colors):
    plott.plot(estfpr[i], esttpr[i], color=color, lw=lw, label='ROC curve of class {0} (AUC = {1:0.2f})'.format(i,
estroc[i]))

# plot the baseline model
plott.plot([0, 1], [0, 1], 'k--', lw=lw)
plott.xlim([0.0, 1.0])
plott.ylim([0.0, 1.05])
plott.xlabel('False Positive Rate')
plott.ylabel('True Positive Rate')
plott.title('ROC Curve of /n Oversampled Decision Tree PCA')
plott.legend(loc="lower right")
plott.show()

# Train and evaluate models
train_scores = []
test_scores = []
for nmkm, Over_trainmo in models:
    model.fit(X_train, y_train)
    train_pred = Over_trainmo.predict(X_train)
    train_scores.append(accuracy_score(y_train, train_pred))
    test_pred = model.predict(X_test)
    test_scores.append(accuracy_score(y_test, test_pred))

# Sort models by test accuracy
sorted_models = [x for _, x in sorted(zip(test_scores, models), reverse=True)]

# Compute cumulative gains
num_samples = len(y_test)
random_gains = np.linspace(0, 1, num_samples)
model_gains = []
for i, (name, model) in enumerate(sorted_models):
    pred_prob = model.predict_proba(X_test)
    order = np.argsort(-pred_prob, axis=1)
    sorted_pred_prob = pred_prob[np.arange(num_samples)[:,None], order]
    class_labels = np.unique(y)
    class_ranks = np.argmax(order == y_test.reshape(-1,1), axis=1)
    class_gains = np.zeros(num_samples)
    for j in range(num_samples):
        class_gains[j] = np.sum(sorted_pred_prob[j][class_ranks[j]+1]) / np.sum(sorted_pred_prob[j])
    model_gains.append(class_gains)

```



```

# Plot gain chart
plott.figure(figsize=(8,6))
plott.plot(nanp.arange(num_samples), random_gains, label='Random Model')
for i, (name, model) in enumerate(sorted_models):
    plott.plot(nanp.arange(num_samples), nanp.cumsum(model_gains[i]), label=name)
plott.xlabel('Number of samples')
plott.ylabel('Cumulative gain')
plott.legend()
plott.show()

model = Over_traimo['OverNeuralNetwork']

testdf = pandd.read_csv('/content/drive/MyDrive/Big-Five_Factors.csv', delimiter='\t')

testdf = testdf[testdf['IPC']==1]
testdf = testdf.dropna()
testdf = testdf.iloc[0:1000,0:50]

testdf.shape

res = pandd.DataFrame(model.predict(testdf))
res.rename(columns = {0:'Class'},inplace = True)
target_classes = list(question_Fivefac['target'].unique())

sens.countplot(x="Class", data=res);
plott.title('Test Data')
plott.show()

```