Endsem part 1
Revision of assignments
Linear Regression

Anik Bhowmick
Inter Disciplinary Dual Degree Data-Science
Indian Institute of Technology Madras
ae20b102@smail.iitm.ac.in

***Abstract*—This part is about a dataset that contains information about cancer incidence and death cases across different states of the US for various population groups. The working of linear regression is demonstrated using this dataset. In addition, various relevant data preprocessing techniques, such as feature selection, data visualizations, linear model fitting and its evaluation. The section that is covered in this revision is The problem, which covers topics from data preprocessing before training to model validation post-training.**

***Index Terms*—Linear Regression, Correlation coefficient, Mean squared error, Residuals.**

I. The Problem

Our goal in this assignment is to predict death cases and mortality due to cancer for various population groups by linear regression with a primary focus on finding a correlation between incidence rate and mortality with socio-economic status.

A. Cleaning and preparing the data

The dataset had some missing as well as categorical entries. In either case, the linear regression model can not work.

1) Handling missing values and categorical columns: Some of the columns of the given dataset had categorical features such as "State" and "AreaName". The "Incidence_Rate" had missing categorical and numerical values all present. We first dropped the first two categorical features. The "_" and "__" of "Incidence_Rate" were dropped as we had no reference with what we should substitute them. These rows belong to certain states like "KS", "MN" and "NV". The same strategy is followed for the other few columns. The missing values of pure numerical columns as "Med_Income", "Med_Income_White", and "Med_Income_Asian", etc., were replaced with the respective column's median values. In the earlier assignment, it was replaced by mean. But it turns out that all the features have skewed distribution. So median is the best strategy.

B. Exploratory analysis

1) Visualizing the correlation among features: To have the overview we obtained the correlation plot first. The formula for correlation between two features x_i and y_i is :

$$r_{xy} = \frac{\sum_i x_i y_i - n\bar{x}\bar{y}}{\sqrt{\sum_i x_i^2 - n\bar{x}^2}\ \sqrt{\sum_i y_i^2 - n\bar{y}^2}}.$$

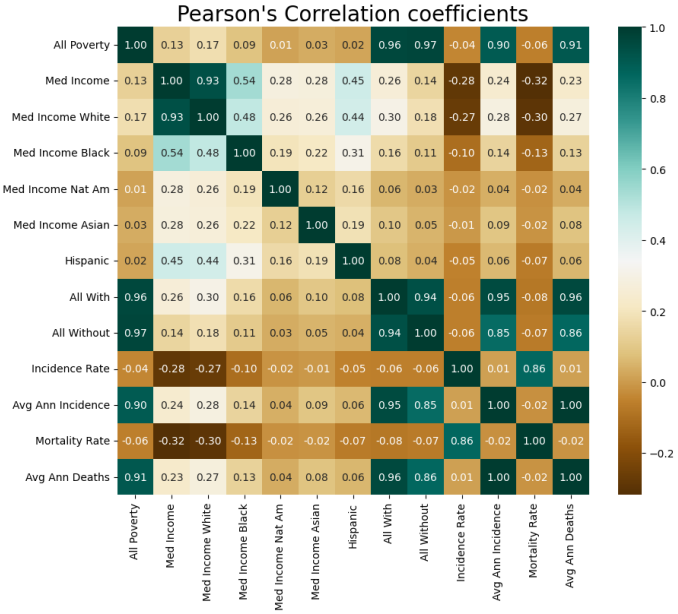


Fig. 1. Correlation plot

C. Outliers detection

Below are box and distribution plots of a few columns

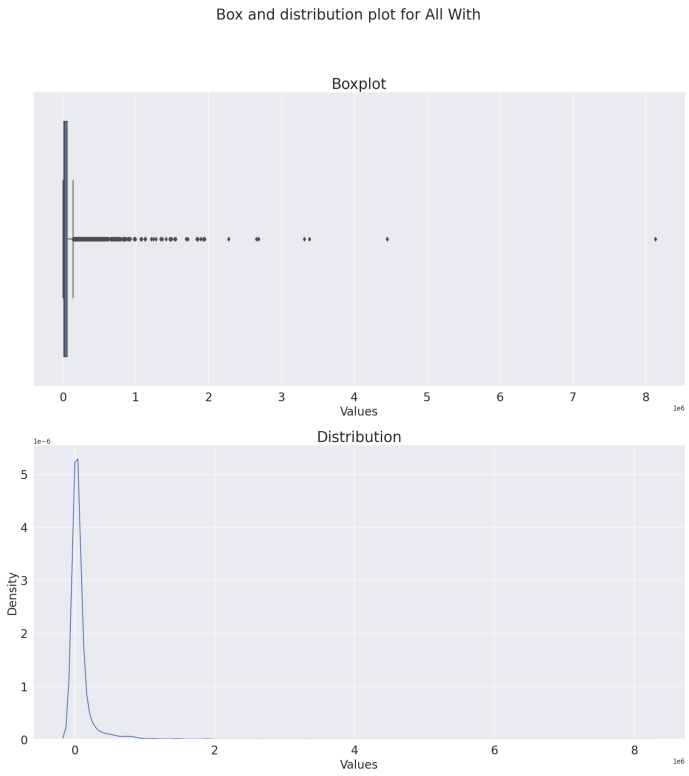


Fig. 2. Box and density plot of all with

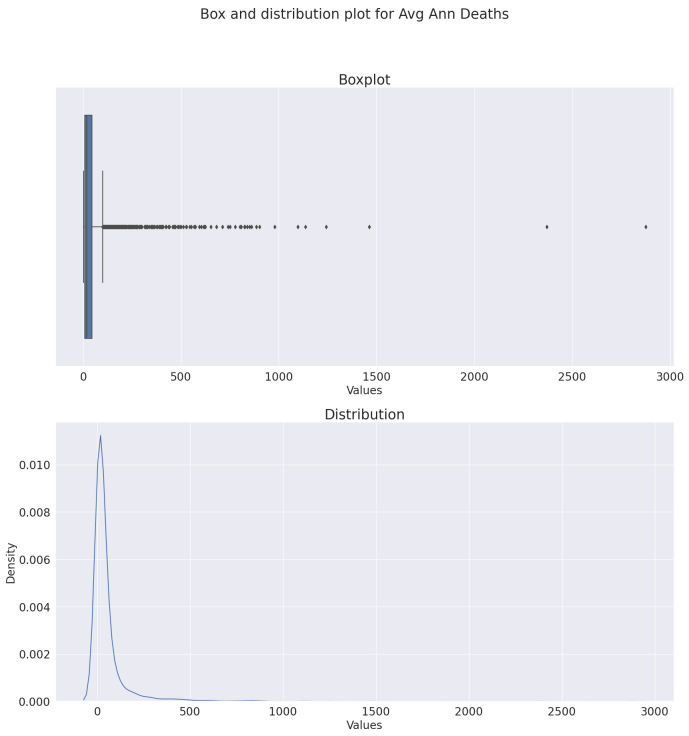


Fig. 4. Box and density plot of Average annual deaths

This way, all the features have outliers. So we need to drop them based on some statistical method. In my case, I calculated the Z-score and dropped the samples with a Z-score higher than 1.5 or lying away from their mean above 1.5σ.

D. Final Correlation plot before training

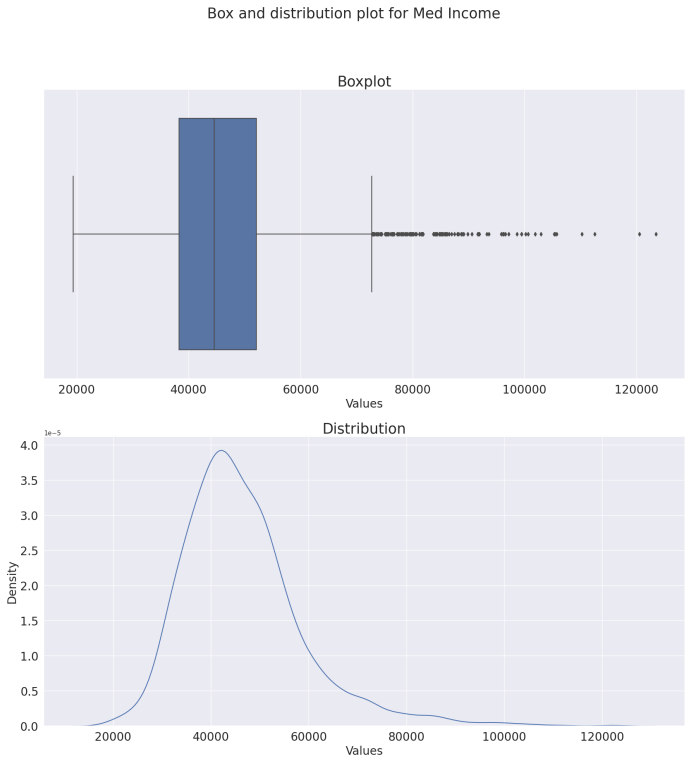


Fig. 3. Box and density plot of medium income

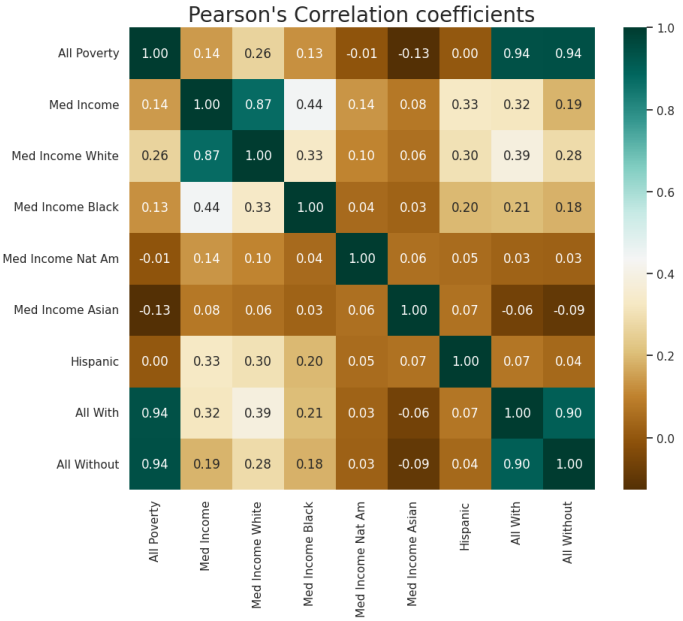


Fig. 5. New correlation plot for features

Now the correlation among the input features has sufficiently reduced. We will further drop highly correlated features. In this case, it is 'All_Poverty'.

E. Statistical model

As we guessed, four target variables are possible, so we decided to make 4 models and check their effectiveness. Further, we used the sklearn library for linear regression to keep the codes simple. Here we will show only the improved models; there are 2 improved models.

1. Model 1 for Avg Ann Deaths

- The new model took 8 input features, resulting in a R^2 score of 0.91. Earlier, it was 0.89. So, it is an improvement because we used the median instead of the mean for imputation. Also, we dropped one highly correlated feature
- The final model was developed to take only one input feature, "All With", because using the feature importance method, we found this feature among all socio-economic features is most important. The model resulted in obtaining the functional form as

$$\hat{y} = 34.99x + 31.5$$

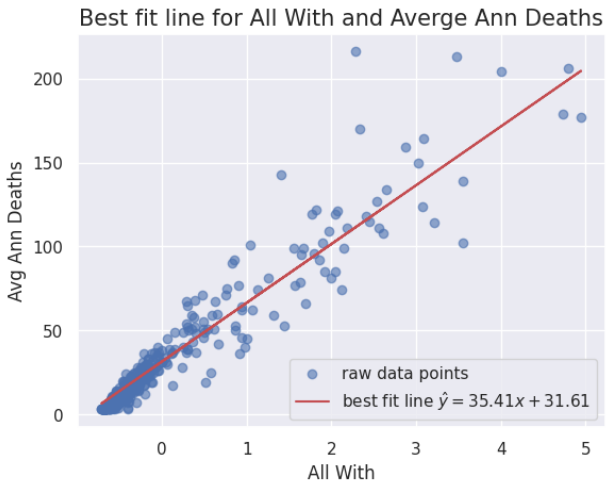


Fig. 6. Best fit for average annual death

1. Model 2 for Avg Ann Incidence

- This new model took 8 input features, resulting in an R^2 score of 0.884. Earlier, it was 0.882. So it is a slight improvement. The simple reason is the median.
- The final model was developed to take only one input feature, "All With", because using the feature importance method, we found this feature among all socio-economic features is most important. The model resulted in obtaining the functional form as

$$\hat{y} = 49.1x + 41.4$$

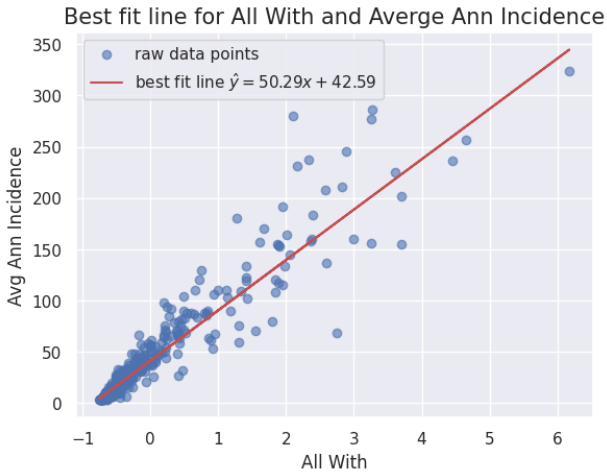


Fig. 7. Best fit for average annual incidence

F. Validation

Fig. 8. Residual plot for average annual death

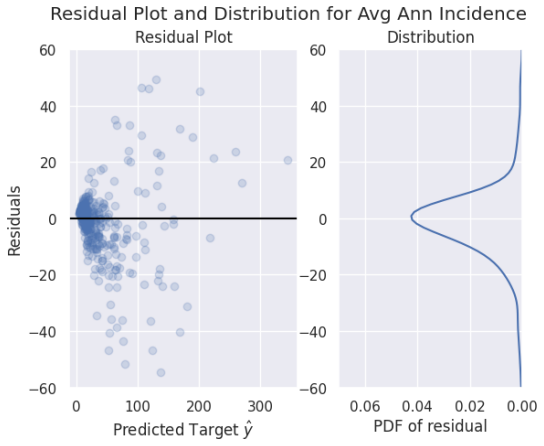


Fig. 9. Residual plot for average annual incidence

The residual plots for these two models suggest that the error is normally distributed around zero, and the variance

is nearly constant. So, Homoscedasticity behavior is present. This implies the model performance is very good.
The other two models for 'Mortality Rate' and 'Incidence Rate' did not show any improvement because these two variables do not depend much on the input features, as can be seen by the first correlation plot. Their R^2 score is very low in the order of 10^{-2}.

G. Answers to the primary goal

- The socioeconomic features primarily include income, education, employment, social status, social privileges, etc. The correlation plots mention all the social features used for model training. The dependence of the targets on these features is illustrated in correlation plots. This correlation plot shows that "Incidence Rate" and "Mortality rate" don't depend much on socioeconomic factors. Average Annual Cancer Incidence and Average Death cases due to cancer are strong functions of socioeconomic factors. So, the Annual Cancer Incidence and Annual Mortality do have high correlations with socioeconomic features. Among all socioeconomic features, the most important one was 'All with'.

II. Conclusions

So, in this revised version of assignment 1, we gained some better model performance by simply tweaking the feature engineering process a little bit. Linear regression is a powerful predictive modelling tool, and this assignment also proved its predictive ability. However, a strong requirement for this purpose is the linear dependence of the target variable on the input variables. That's why even with the modification in the feature engineering process, we did not get any improvement in the 'Mortality Rate' and 'Incidence Rate'.

References

[1] Linear Regression: https://www.geeksforgeeks.org/ml-linear-regression/
[2] Gradient Descent: https://builtin.com/data-science/gradient-descent
[3] Evaluation Metrics: https://www.analyticsvidhya.com/blog/2021/05/know-the-best-evaluation-metrics-for-your-regression-model/
[4] Residual plot: https://towardsdatascience.com/how-to-use-residual-plots-for-regression-model-validation-c3c70e8ab378
[5] Feature engineering: https://medium.com/analytics-vidhya/feature-engineering-part-1-imputation-techniques-eafce8f341bc
[6] scikit-learn: https://scikit-learn.org/stable/modules/linear_model.html

Logistic regression

Anik Bhowmick
Inter Disciplinary Dual Degree Data-Science
Indian Institute of Technology Madras
ae20b102@smail.iitm.ac.in

***Abstract*—This assignment discusses the application of a classification algorithm named Logistic Regression on the famous Titanic dataset. On April 15, 1912, the Titanic sank after colliding with an iceberg on its first voyage. More than 50% people died in this unfortunate incident. However, some groups of people survived. This is a revision task of the already trained model, hoping to increase the performance, plots, explainability, etc. In this revised version, the problem section of the original paper will be touched.**

***Index Terms*—Feature selection, visualization, Logistic Regression, Sigmoid Function, Confusion Matrix, Accuracy, Precision, Recall, F_1 score, ROC.**

I. The Problem

This assignment aims to predict whether a passenger survived the Titanic accident or not by logistic regression. The data preprocessing is part almost similar to the assignment one. We tried to improve the model with the help of cross-validation.

A. Cleaning and preparing the data

The dataset had some missing and categorical entries. In either case, the logistic regression model can not work.

1) Handling the categorical columns: Some of the columns of the given dataset had categorical features such as "Sex", "Embarked", and "PClass". In Pclass, it is actually ordinal, so we have to consider it as categorical data. Because the model does not know it is ordinal data. It is just some number to it. But it can give more importance to larger numbers than smaller numbers or vice versa. To avoid this ambiguity, we will convert this also to one hot encoded vector. The table below explains how one hot encoding really works.

Emabarked	Embarked_C	Embarked_Q	Embarked_S
S	0	0	1
C	1	0	0
S	0	0	1
S	0	0	1
S	0	0	1

TABLE I
One hot encoded feature vector for Embarked column for three ports

So, we see for the Emabarked S case, we get a vector representation (0,0,1). For C, it is (1,0,0), and for Q, it is (0,1,0). This kind of data representation is one hot encoded representation of the Emarked column. The same goes for Sex and PClass.

2) Merging some columns to single column: We see that survival is comparatively higher for people travelling with families having 1-2 family members. At the same time, passengers travelling alone died more in number.

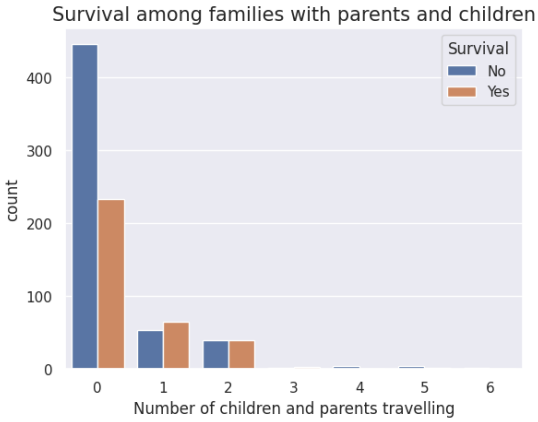


Fig. 1. Passengers travelling with parents and children

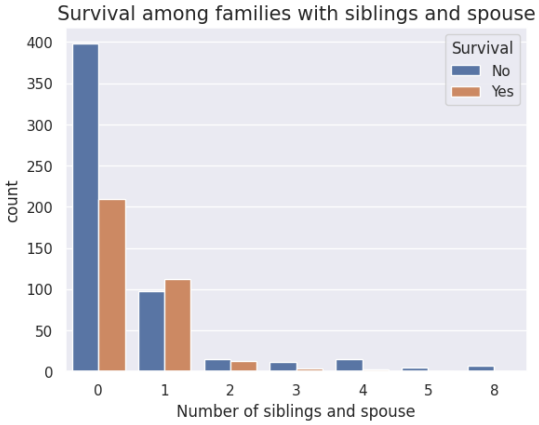


Fig. 2. Passengers travelling with siblings and spouse

So, instead of considering these two columns separately, we can merge them into a single column with the name 'Number of Family members'. The final bar plot looks as below.

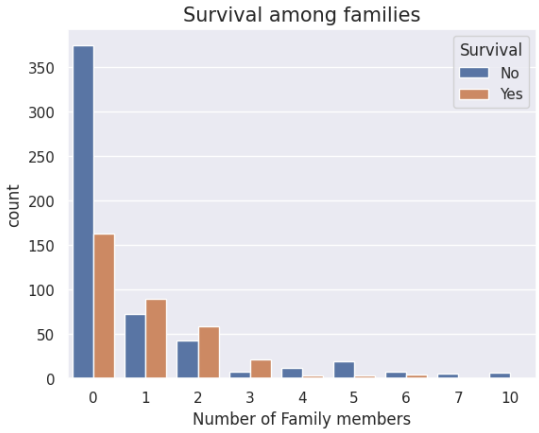


Fig. 3. Merged column

This plot shows that passengers with 1-3 family members survived more than others.

3) Handling the Missing values: As already mentioned, the "Age", "Cabin", and "Embarked column" has missing entries.

- In the **"Embarked"** column, we replaced the missing value with the port having the highest number of passengers boarded. In this case, it is Southampton.

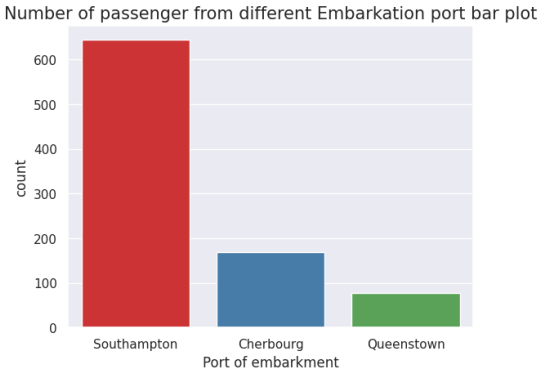


Fig. 4. Bar plot showing the number of passengers embarked from three ports

- For **"Age"** column, when we visualized the data distribution, it turned out to be right skewed. In the case of skewed data, we impute the missing values with the median.

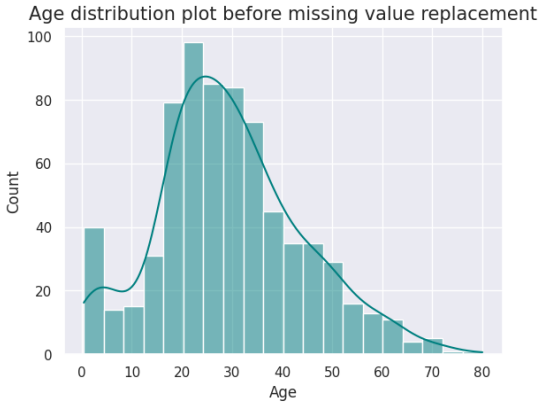


Fig. 5. Distribution of Age

Just like the assignment here, we have used KNN imputer without the target class. The distribution with the fitted curve is given below.

Fig. 6. Distribution of Age after missing value imputation

- In **"Cabin"** column, more than 80% being missing we decided to drop the column entirely. Because we won't be able to extract any useful information from this feature.

B. Exploratory analysis

1) Visualizing the correlation among features: Features are almost uncorrelated, as shown in the correlation plot :

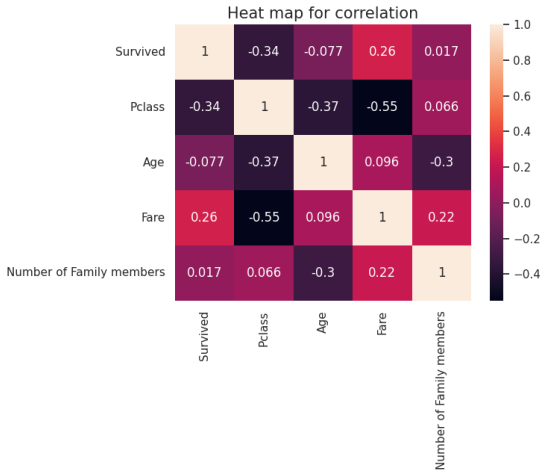


Fig. 7. Correlation plot

C. Splitting the Training data to Train and validation sets

The entire data is split into train and test sets with a test split size of 0.3. Further, we performed cross-validation with K = 10 fold validation.

D. Statistical model

Our model is a logistic regressor from sklearn tuned with hyperparameters. We used GridSearch cross-validation having K Fold validation = 10. GridSearch CV is a cross-validation technique used for hyperparameter tuning. With a given number of parameters, it finds out which parameters are responsible for the best model score. In situations where the dataset is small, cross-validation becomes even more important. With limited data, obtaining reliable performance estimates with a single train-test split is challenging. Cross-validation allows us to make the most of the available data by repeatedly training the model on different folds. This helps in the reduction of both bias and variance. The model parameters are given below.

$$z = 2.30207894 - 1.72947805x_1 + 0.21167606x_2$$
$$-1.34677625x_3 + 0.9596724x_4 + 0.22759791x_5$$
$$-1.18688561x_6 + 0.19601209x_7 + 0.15405493x_8$$
$$-0.34968233x_9 - 2.35825682x_{10}$$

$$h(x;\boldsymbol{\theta}) = \frac{1}{1+e^{-z}}$$

We used two regularizations, L_1 and L_2. Of these, GridSearchCV selected L_2 with the maximum number of iterations =50.

E. Visualization and validation

Below are the evaluation metrics for the validation set: In the assignment 2, we had a test split of 0.15. On that test set, the obtained F_1 score was 0.88 for the death class and 0.76 for the survived class. Here, in the new model, similar numbers are obtained with a larger test size of 0.3. Undoubtedly, this model is doing far better by introducing more generalization into it.

Survival	Precision	Recall	F1-score
0	0.83	0.90	0.87
1	0.82	0.71	0.76

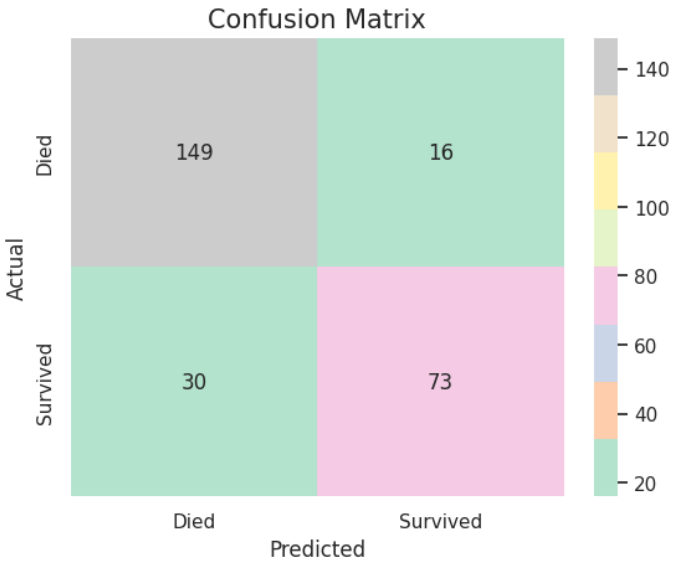


Fig. 8. Confusion Matrix

From this figure, we can infer the following

- **True Positive**=73 (Actual survived, predicted survived)
- **False Positive**=16 (Actual died, predicted survived)
- **False Negetive**=30 (Actual survived, predicted died)
- **True Negative**=149 (Actual died, predicted died)

From this, we get the accuracy as:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$
$$\text{Accuracy} = \frac{73+149}{73+149+16+30} = 0.828$$

The goodness of a model can be tested by one more metric called Area under the ROC (Receiver operating characteristics) curve. The more the area, the better the model is. In our case, we achieved an AU-ROC of 0.86. Which indicates the model is performing well.

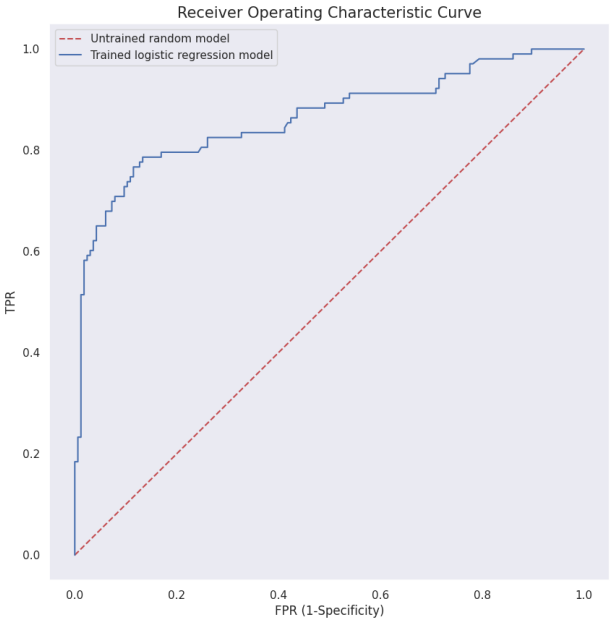


Fig. 9. Receiver operating characteristic curve

F. Insights captured by studying the data

After going through the training data, we captured some patterns. They are discussed below.

1) Insights in test data:

- The training data is a little imbalanced, with more deaths than survivals. The plot given below gives that comparison.

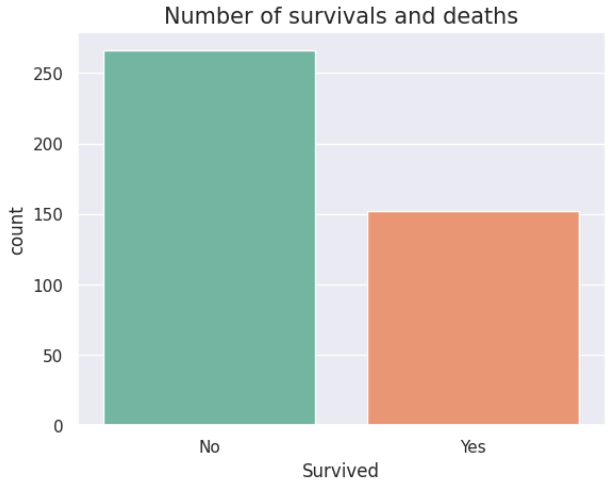


Fig. 10. Survival data

- Among males and females, the females were given more priority for survival.

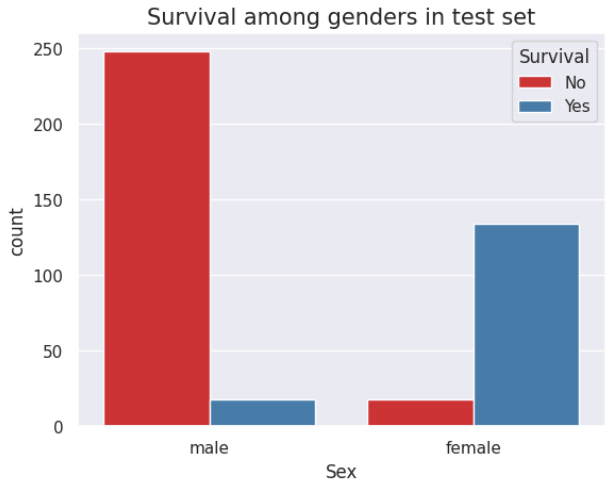


Fig. 11. Survival rate among genders

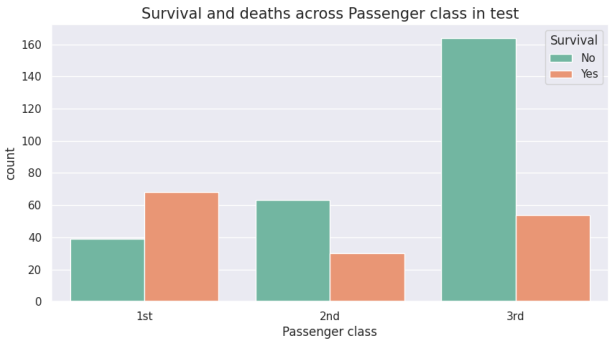


Fig. 12. Survival rate among passenger classes

- We further found that elite classes were prioritised for survival. So the survival rate among 1st class passengers is higher. Maximum number of death cases are from the 3rd class.

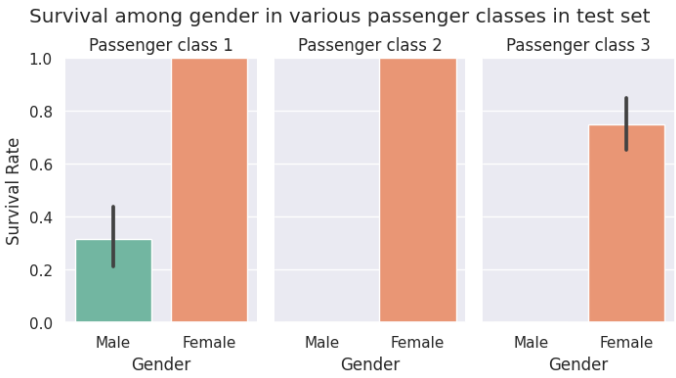


Fig. 13. Survival rate among genders in passenger classes

- From the above figure, it is clear females survived more than males across all passenger classes.
- We also checked the distribution of survival cases across various age groups. We found out that although death

cases are almost equal for all age groups, younger people, especially those under 10, have a slightly higher survival rate.

Fig. 14. Survival rate among various age groups

So, by observing the data, we can infer the following in terms of survival:

- Females and children were given higher priority for survival.
- Passengers travelling in the first class survived more due to the fact that they were a highly privileged group. In fact, a large fraction of the male survival came from the 1st class only.

II. Conclusions

So, in this revised version of assignment 2, we gained some better model performance by simply changing the training procedure with the help of GridSearchCV. The GridSearchCV helps mitigate the data scarcity. The training set is quite small in size. Without the cross-validation technique, the model training won't be correct. It mostly overfits the training data and fails to generalize on the test set. With proper hyperparameter tuning, the logistic regression model can be prevented from overfitting. So this revised version successfully demonstrated the applicability of Gridsearch cross-validation with a slight improvement in model performance on the Titanic dataset.

References

[1] Logistic Regression: https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc
[2] Evaluation Metrics: https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/
[3] Scikit learn KNN imputer: https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html
[4] Scikit learn logistic regression: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[5] Scikit learn GridSearchCV: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Naive Bayes classifier

Anik Bhowmick
Inter Disciplinary Dual Degree Data-Science
Indian Institute of Technology Madras
ae20b102@smail.iitm.ac.in

***Abstract*—This assignment discusses the application of a Naive Bayes classifier on the Adult dataset. This data was extracted from the 1994 Census Bureau database by Ronny Kohavi and Barry Becker. This data indicates whether a person has an annual income of more or less than 50,000 US dollars based on various attributes like age, gender, education qualification, working class, work hours, and native country. By studying the data and fitting the model, our task is to unfold which kinds of people are in higher income groups and which are less. The Problems section will be highlighted for the model performance improvement in this revised version of the assignment.**

***Index Terms*—Visualization, K Modes, Naive Bayes classifier, Voting classifier, One hot encoding, ComplementNB, CategoricalNB, BernoulliNB, Confusion Matrix, Accuracy, Precision, Recall, F_1 score, pipeline, ROC.**

I. The Problem

As mentioned, our goal in this assignment is to predict whether a person earns above 50k dollars by 3 different Naive Bayes models. We followed the following steps to prepare the data before feeding it into the model.

A. Cleaning and preparing the data

The dataset has almost entirely categorical data in the form of strings.

1) Handling Missing values: After thoroughly studying, we found that the categorical columns containing missing values are: 'Workclass', 'Occupation', and 'Native_country'. They have '?' in the place of missing values. We used an unsupervised algorithm K-prototypes to impute these missing values. It is a clustering technique that works for both numerical and categorical columns. For numerical missing values, we can assign respective cluster mean, and for categorical missing values, we can use cluster modes. We won't discuss the details of this process in this assignment. Below is a brief description of our missing value imputation technique.

- First, drop all the columns which have missing values. Now, with the remaining data, fit the k prototype with the suitable number of clusters.
- Once the clusters are formed, check for each missing column data to which cluster it belongs. Once that is found, use the mode of that particular cluster for that particular column to replace the missing entry.

In my case, I experimented with several clusters. I found 4 clusters gave better model results. So, I chose 4 clusters.

2) Encoding the categorical columns: Because any ML model can't work with strings, we need numerical representations of the categorical features. For this purpose, we decided to use one hot encoding. One hot encoding converts the categorical data to vectors containing zeros and ones. For example, when we apply one hot encoding to the gender column, it will make vectors (1,0) and (0,1). The first vector may denote male, and the second one will be female or vice versa. Here, we got two components of each vector because the gender column has only two kinds of instances: male and female. If any column has n different instances, we will have n different one-hot vectors, each having n components.

B. Exploratory analysis

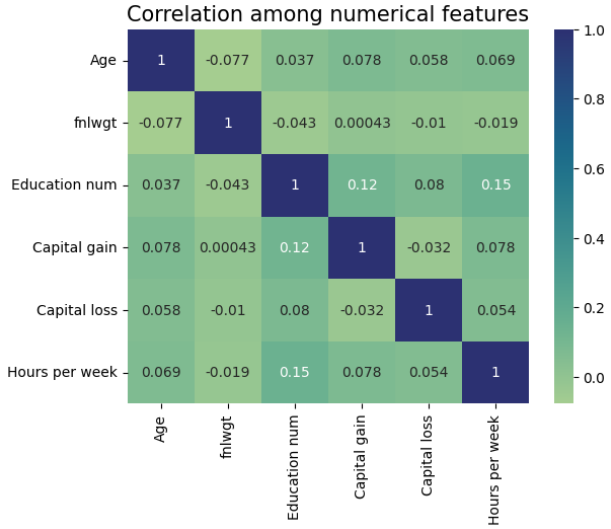


Fig. 1. Correlation plot

1) Correlation among the numerical featurers: Two features are correlated if they have an absolute correlation coefficient close to 1. They are uncorrelated if their correlation coefficient is closer to 0. If a model is fed with correlated data, model training will be unstable, and overfitting may become predominant. So, the usual technique to get rid of correlation is to drop the highly correlated features. The formula for

correlation between two features x_i and y_i is :

$$r_{xy} = \frac{\sum_i x_i y_i - n\bar{x}\bar{y}}{\sqrt{\sum_i x_i^2 - n\bar{x}^2}\ \sqrt{\sum_i y_i^2 - n\bar{y}^2}}.$$

But in our case, all the numerical features are almost uncorrelated. Because the values are quite close to zero, this is really helpful for us, as we don't need to drop any numerical features. The Naive Bayes classifier will work better on such data because it considers all the features equally important. In the case of categorical features, because we will encode them by one hot encoding technique, they will be uncorrelated.

2) Distribution Plots: It's sometimes useful to check the distribution of numerical features. As distribution sometimes gives very useful insights.

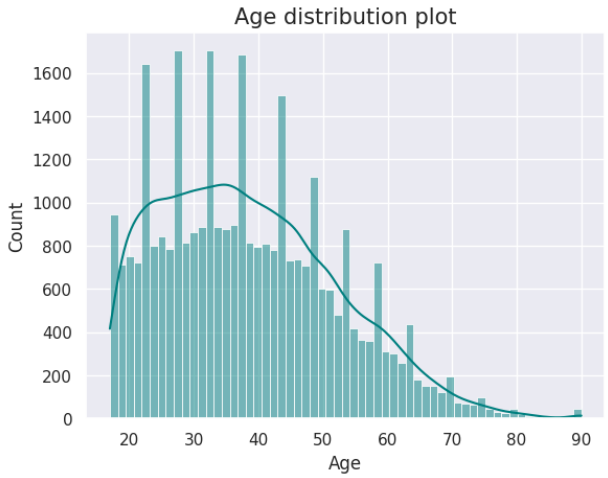


Fig. 2. Distribution of age

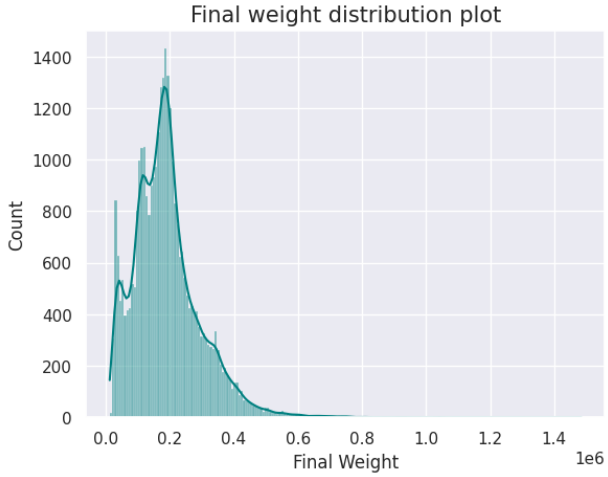


Fig. 3. Distribution of Final Weight (fnlwgt)

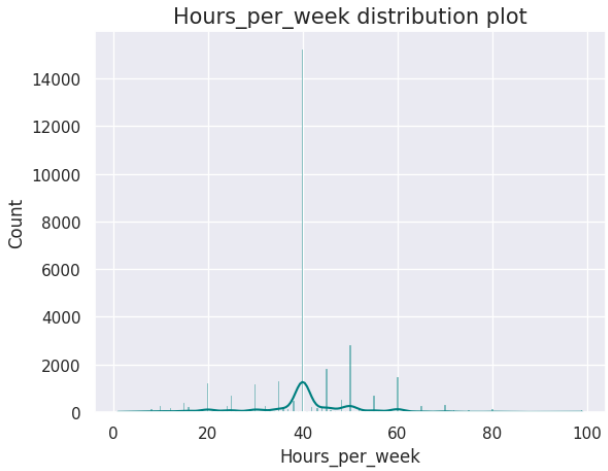


Fig. 4. Distribution of hours per week

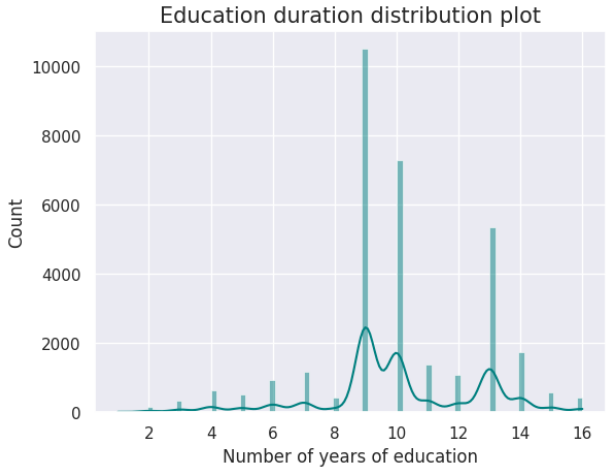


Fig. 5. Distribution of duration of education (years)

From these distribution plots, we see except for the age and final weight columns, all other columns have an uneven distribution of data (Some have very high occurrence, some are very low). This may lead to erroneous model training, so we treated numerical columns as categorical. In this case, the model won't prioritize major data points compared to minor ones.

3) Preliminary study on the trend followed by data: We will study a few plots before finally feeding the data to the model.

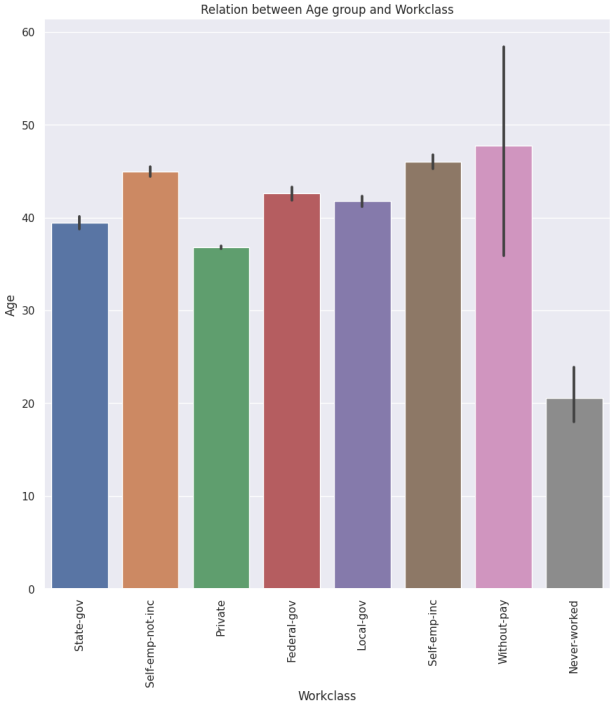


Fig. 6. Works class vs Age

From this plot, we see younger people under 20 never worked. It is as per expectation.

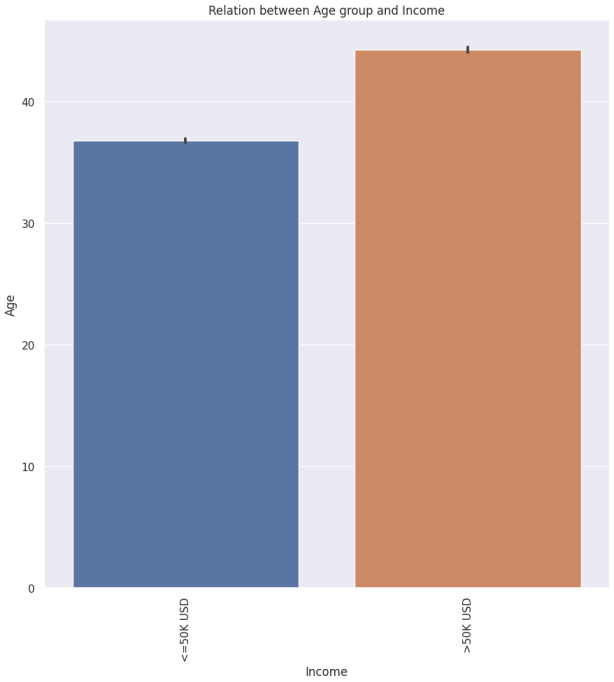


Fig. 7. Age and Income relation

All the age groups earning above 50K USD are above 40 years old. This may be because, with increasing age, working experience and exposure increase. So salary can also be expected to increase.

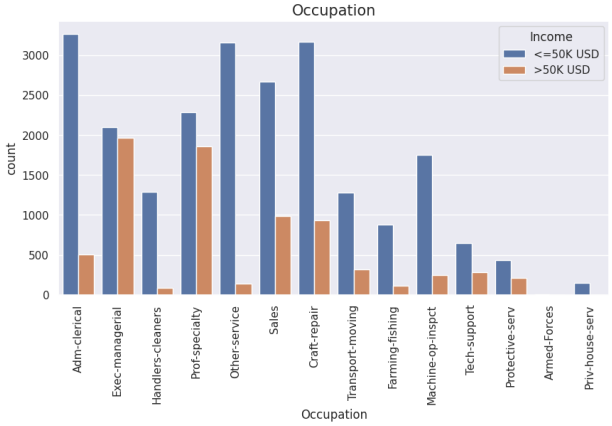


Fig. 8. Occupation count

We see people from executive and managerial occupations are more likely to earn above 50K USD. Whereas clerics, craft repairers, and other job workers are lower-income groups. Our common knowledge also tells us that executives make more money than others.

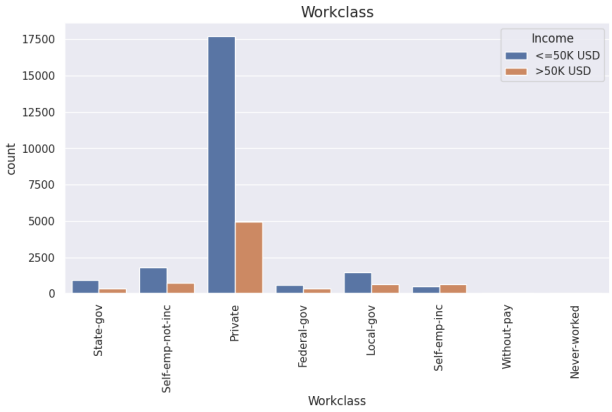


Fig. 9. Work place count

This plot reveals that people in private firms are more likely to earn above 50K USD.
The sole purpose of this kind of preliminary study is to check whether the model also follows this kind of trend after training on the validation set.

4) Visualizing the class distribution: We used a bar plot to get an idea of the number of data points greater and less than 50,000 dollars.

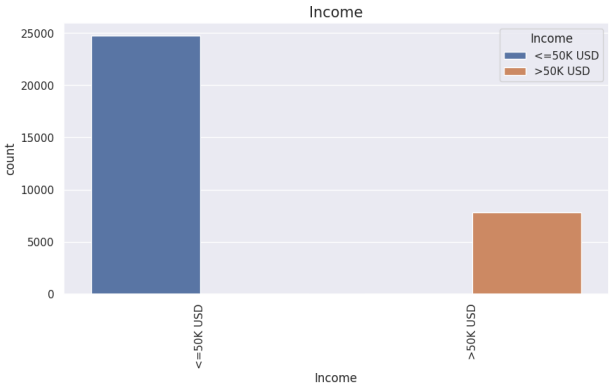


Fig. 10. Income count plot

There are 24720 people with less than 50K dollars in income and 7841 people with greater than 50K dollars. Undoubtedly, the data is highly imbalanced. During the train test split, we have to maintain this ratio of the number of data points of each class. For this purpose, we have to stratify the splitting strategy. Techniques like under-sampling and oversampling also might help. But we will keep our task simpler and easily understandable so we won't adopt those techniques.

C. Splitting the Training data to Train and validation sets

To validate the model, we decided to split the model into training and validation sets. Because the number of data points is fairly large, a 65:35 splitting ratio will work fine. The test point has been increased than the previous version, considering the robustness of the model.

D. Statistical model

Our model is an Ensemble voting classifier from sklearn. This classifier ensembles three different kinds of Naive Bayes classifier compatible to work with categorical features. These are ComplementNB, CategoricalNB, and BernoulliNB. The first one is developed to work with highly imbalanced data. The other two work really well with categorical boolean features. So basically, we merged the working of three classifiers via a pipeline necessary for data preprocessing. The model cross-validated accuracy for the train set was found to be 84%, which is quite decent. The result of the model on the larger test set also appeared to be superior, with 84% accuracy. In the earlier assignment version, we had almost the same accuracy (83%), but at that time, the test set was quite smaller compared to the current one.

E. Visualization and validation

Below are the evaluation metrics for the validation set: Our

Income	Precision	Recall	F1-score
<=50K USD	0.91	0.88	0.89
>50K USD	0.65	0.72	0.68

precision and recalls have increased slightly compared to the earlier one.

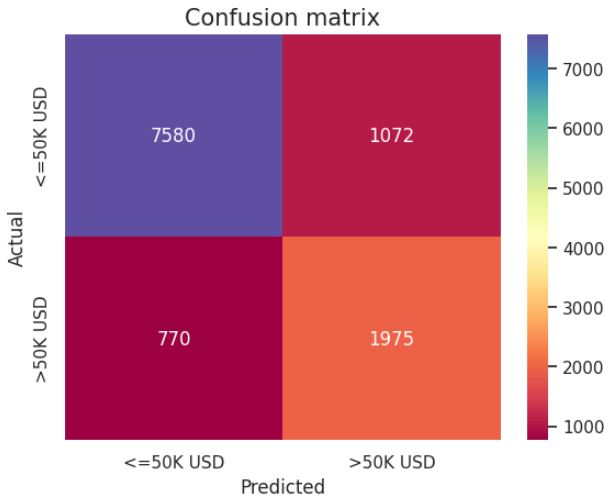


Fig. 11. Confusion Matrix

From this figure, we can infer the following

- **True Positive**=1975 (Actual above 50K USD, predicted above 50K USD)
- **False Positive**=1072 (Actual below 50K USD, predicted above 50K USD)
- **False Negetive**=770 (Actual above 50K USD, predicted below 50K USD)
- **True Negative**=7580 (Actual below 50K USD, predicted below 50K USD)

From this, we get the accuracy as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Accuracy} = \frac{7580 + 1975}{7580 + 1975 + 1072 + 770} = 0.84$$

The goodness of a model can be tested by one more metric called Area under the ROC (Receiver operating characteristics) curve. The more the area, the better the model is. In our case, we achieved an AU-ROC of 0.91. Which indicates the model is performing quite well.

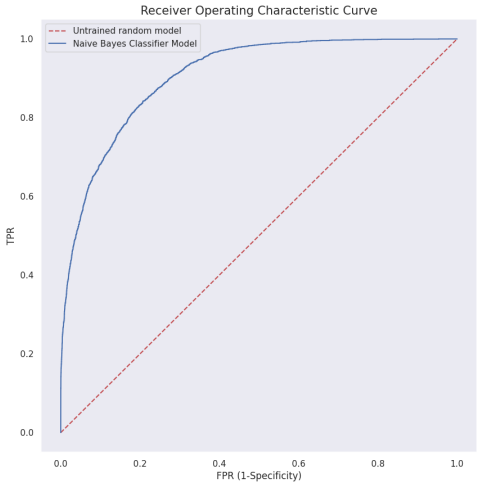


Fig. 12. Receiver operating characteristic curve

F. Insights captured by studying the data

After going through the training data, we captured some patterns. They are discussed below.

1) Insights in training data:

- From the education plot, we see that the largest income group above 50K USD is Bachelors. In the case of master's degree holders, professors and doctorates, more people earn above 50K USD than below 50K USD. The lowest income groups are school pass-outs. So evidently, education qualification matters in deciding income.

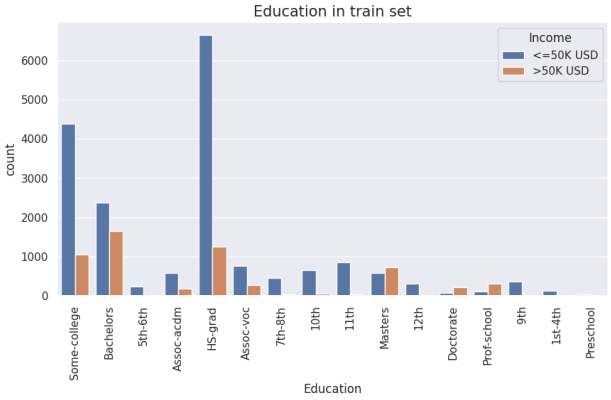


Fig. 13. Education in train set

- This figure suggests most higher income groups are married. Whereas lower-income groups are mainly unmarried, divorced or separated. This clearly conveys the message that income really influences marriage life.

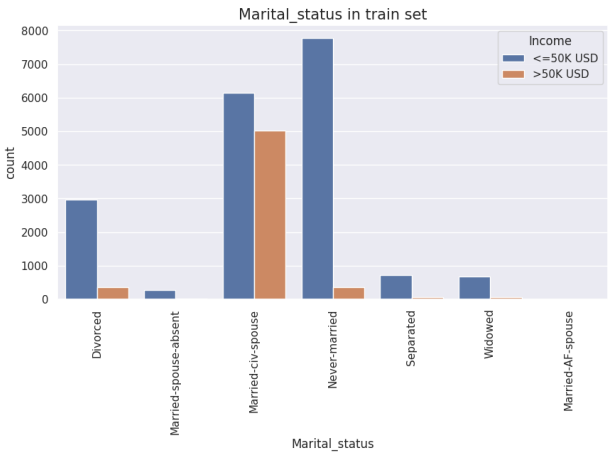


Fig. 14. Marital status and income count

- Already in the exploratory analysis, we mentioned that Executives and managers are the highest income groups. In the train set, we see the same too. Low-income groups mostly belong to clerics, other job workers and craft repairers.

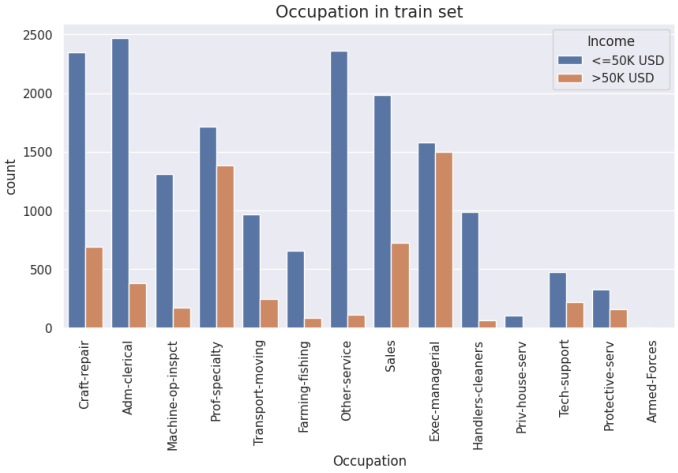


Fig. 15. Occupation count in train set

- This plot gives us information regarding the relation between income and a person's race. In this case, it's somewhat difficult to draw an inference; although the highest income group is white, the largest number of lower income groups also belong to the white group. And the difference between these two is quite high.

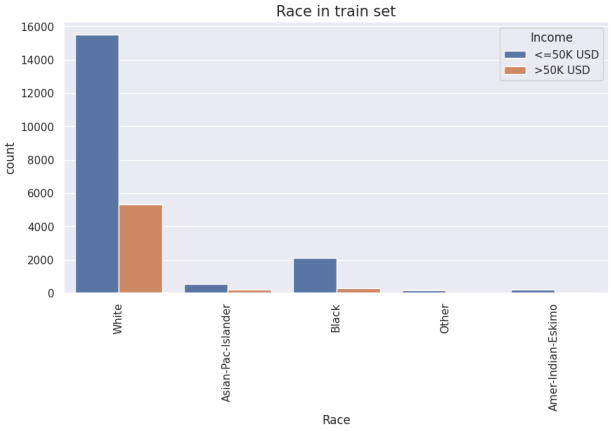


Fig. 16. Train set Race count

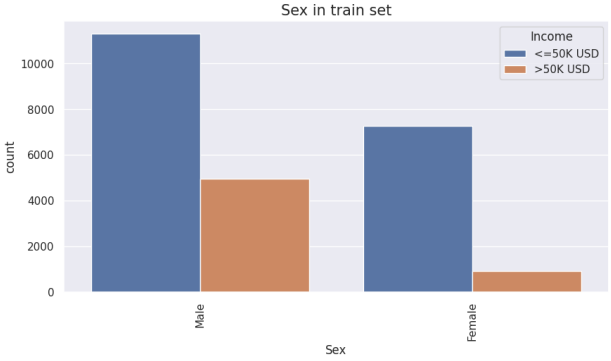


Fig. 17. Gender and income relation in train

- Males are more likely to earn money than females. The higher income group is also from the male gender.

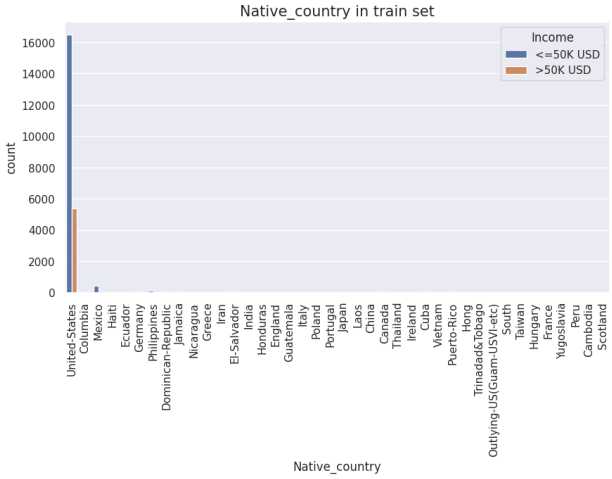


Fig. 18. Origin country and income relation

- People from the United States are more income groups. For other countries that didn't even appear in the plot, most have count values below 100. That's why they are almost invisible in the plot compared to the count from the US.

2) Insights in testing data: We expect similar patterns in the predicted test data set. This will ensure the model is behaving consistently both on seen data as well as on unseen data.

Fig. 19. Income count in test set.

- The distribution is the same, i.e., more people are in the lower-income group.
- Here also we see Bachelors, Masters and Doctorates are higher income groups. This is quite reasonable because, as per expectation, pay increases with education qualification.

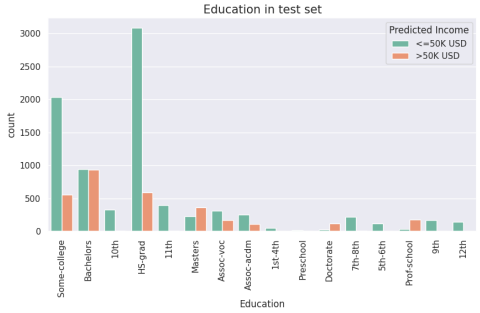


Fig. 20. Test set education distribution with income.

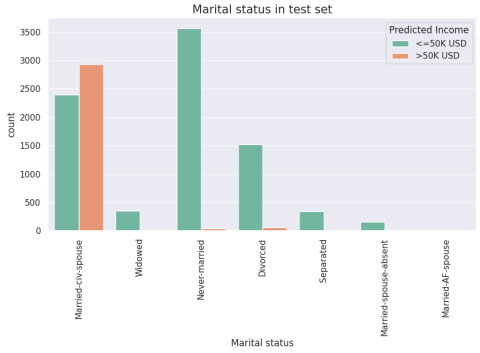


Fig. 21. Marital status on the test set

- Again, here, too, people who earn more have a higher frequency of marriage. The largest lower-income groups are those who never married; it may be possible that these people are school/ college students. Or maybe people who have lower earnings never opt for marriage. So overall, we see a similar structure in the model predicted set also. These are clear indications of the good performance of the model.

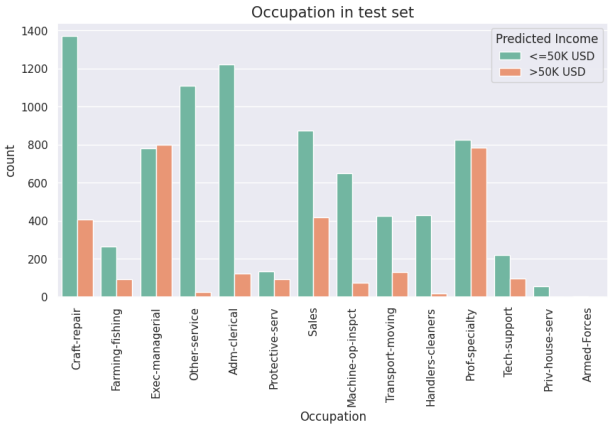


Fig. 22. Occupation in the test set

- Here, the high-income group is from executive and managerial posts. Further, a new occupation emerged as the highest income group Prof-speciality.

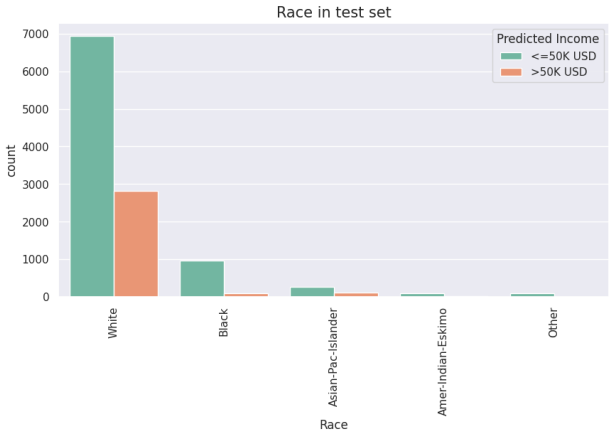


Fig. 23. Race in test set.

- From this plot, we are unable to draw a satisfactory conclusion. It is quite obvious that a person's race/ background should not affect his income potential. The income potential should be solely dependent on the education qualification, experience and exposure in a bias-free society.

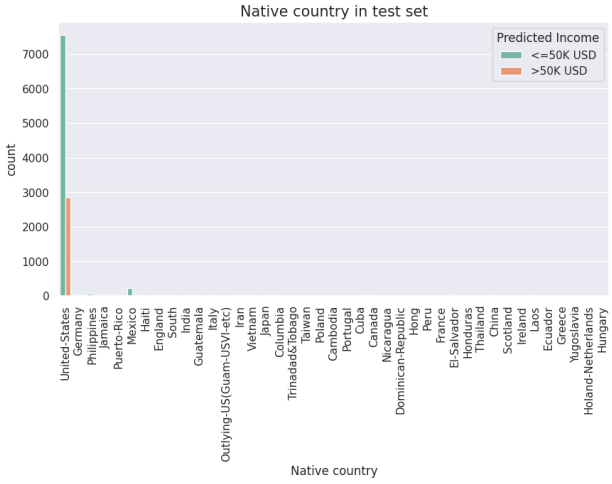


Fig. 24. Country vs income distribution in test set.

- This country distribution plot also looks the same as the training one, with people from the US being the largest in number for earning groups.

So, by observing the training and testing data, we can infer the following in terms of annual earnings:

- Education qualification plays a key role in deciding someone's earning potential.
- Executives and managers in private companies, professors and doctorates in education earn more.
- A person's earning really influences his/ her social life and relationships with others.

So, whatever inference we drew so far is from the train and model predicted test set, the behaviour of the predicted test set is quite similar to the train set ground truths. So, instead of relying only on the result of classification metrics, we cross-validated the entire model performance on our own, which gives good confidence in the working ability and explainability of the model.

II. Conclusions

Naive Bayes is the simplest and easiest algorithm to implement among various classification algorithms. The property that makes it unique from most other classifiers is that it is a generative model, meaning it generates the distribution from which the data is drawn, whereas other discriminative models don't generate any distribution. There are certain limitations in the Naive Bayes, such as it assumes linear independence between the input variables, which is wrong in most real-world cases. In this revised version of the Naive Bayes assignment, we have successfully implemented the voting classifier using 3 different Naive Bayes models. That leveraged the unique capacity of 3 individual Naive Bayes classifiers, which helped us improve the accuracy of the model.

References

[1] Naive Bayes: https://www.geeksforgeeks.org/naive-bayes-classifiers/
[2] Evaluation Metrics: https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/
[3] Pandas: https://pandas.pydata.org/
[4] K modes: https://www.geeksforgeeks.org/k-mode-clustering-in-python/
[5] K prototypes: https://kprototypes.readthedocs.io/en/latest/api.html
[6] Scikit learn complement naive bayes: http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.html
[7] Scikit learn categorical naive bayes: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.CategoricalNB.html
[8] Scikit learn bernoulli naive bayes: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html
[9] Seaborn: https://seaborn.pydata.org/index.html

Decision Tree

Anik Bhowmick
Inter Disciplinary Dual Degree Data-Science
Indian Institute of Technology Madras
ae20b102@smail.iitm.ac.in

***Abstract*—This assignment discusses the application of the Decision Tree Classifier on the Car Evaluation Data Set. This data was derived from a simple hierarchical decision model originally developed for the demonstration of DEX, M. Bohanec, V. Rajkovic: Expert system for decision making. Sistemica 1(1), pp. 145-157, 1990.). The data comprises various kinds of car features, such as the buying price, maintenance price, maximum passenger capacity, safety, etc. Based on it, the task is to predict whether a car is acceptable or not, depending on its safety. This revised version of assignment 4 aims to improve model performance. For that reason, we will discuss the problem section here.**

***Index Terms*—Visualization, Decision tree classifier, Label encoding, Cramer's V rule, Correlation coefficient, Confusion Matrix, Accuracy, Precision, Recall, $\mathbf{F_1}$ score, ROC.**

I. The Problem

In this revised section, our goal in this assignment is to predict whether a car is safe, acceptable, or unacceptable with slightly better model performance.

A. Cleaning and preparing the data

1) Encoding the categorical columns: Because any ML model can't work with strings, we need numerical representations of the categorical features. There are two kinds of encoding primarily used in Machine learning tasks. One hot encoding and Label encoding. In our case, we decided to continue with the label encoding technique. One advantage is that it does not unnecessarily increase the number of features corresponding to each unique instance. For a small dataset like this, an increased number of features might lead to overfitting of the model.

B. Exploratory analysis

1) Correlation among the numerical featurers: Two features are correlated if they have an absolute correlation coefficient close to 1. They are uncorrelated if their correlation coefficient is closer to 0. For categorical features, the conventional person correlation technique can not be applied. There is a technique called Cramer's V correlation is widely used for this purpose. It ranges from 0 to 1, indicating no association to a perfect association, respectively. First, the contingency table is calculated. It lists frequencies or counts of the combinations of every two categorical variables. Then the chi-squared test is performed.

$$\chi^2 = \sum \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

Here O_{ij} is the observed frequency in each cell, and E_{ij} is the expected frequency in each cell. The Cramer's V rule is given as :

$$V = \sqrt{\frac{\chi^2}{n \times \min(k-1, r-1)}}$$

Where n is the total number of observations, k is the number of categories in a variable, and r is the number of categories in another variable.

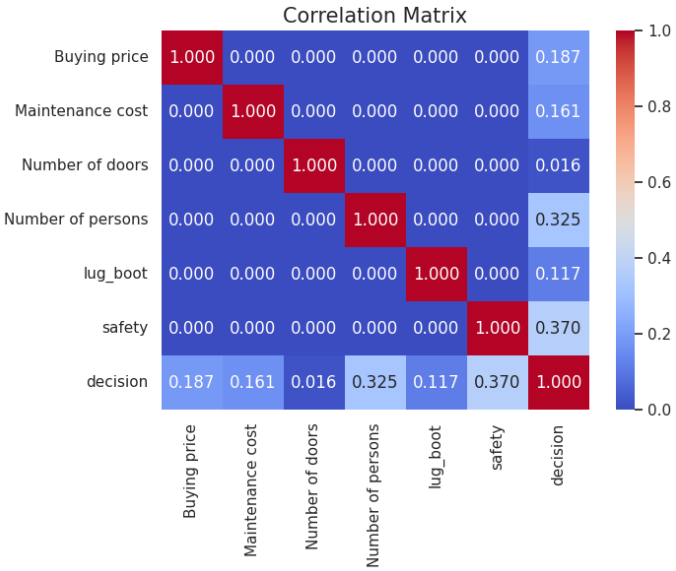


Fig. 1. Correlation plot

All the input categorical features are uncorrelated because the values of correlation coefficients are zero. So, there is no need to drop any column.

2) Preliminary study on the trend followed by data: We will study a few plots before finally feeding the data to the model.

Fig. 2. Buying price and acceptability

From this plot, it is clear that expensive cars have lesser acceptability. Whereas cheaper cars are more under acceptable, good, and very good class. The number of unacceptable cars is much less than that of expensive cars.

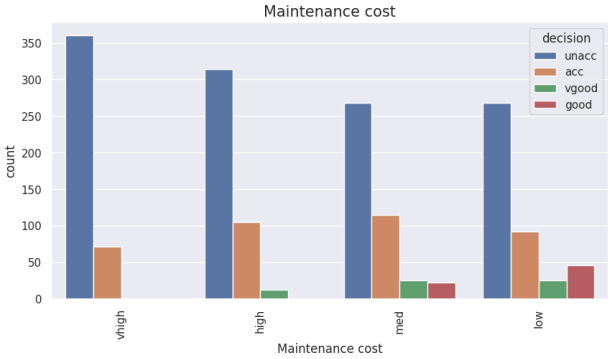


Fig. 3. Acceptabilty with maintenance cost

Cars with low maintenance costs are rejected comparatively less than those that take huge maintenance costs. It is evident from this plot.

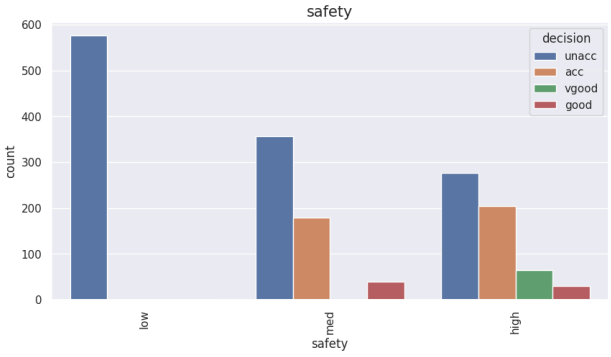


Fig. 4. Acceptance with car safety

Common intuition suggests that, safer the car, the more it will be accepted. And unsafe cars will be less acceptable. The same is reflected in the above plot. Here, unsafe cars have no acceptability at all.

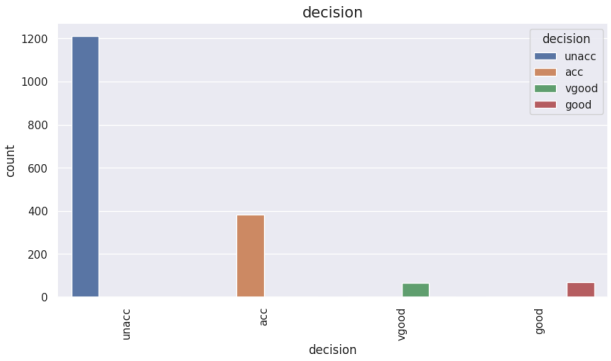


Fig. 5. Absolute count of 4 classes

This plot clearly conveys the fact that classes are highly imbalanced, with the unacceptable class being the largest in number. This gives a clear indication that accuracy can not be used for the model validation. Other metrics, such as precision, recall and F_1 score, need to be used. The true counts of these 4 classes are unacc: 1210, acc: 384, good: 69, vgood: 65.

C. Splitting the Training data to Train and validation sets

To validate the model, the data will be split into training and test sets, with test size being 20%. Because of the imbalanced dataset, it is highly necessary to make stratified splits. Because it is never desirable that the training set does not contain any minority classes viz, good and vgood at all, and all the good, vgood classes are only present in the test set. Naturally, the model outputs will be full of errors, and the model might not even see the minority classes during training. So, stratification is very important for this dataset. This ensures both the train and test set contain all the classes with almost similar ratios. In the revised model, the model is trained with cross-validation of the train set.

D. Statistical model

Our model is a Decision tree classifier from sklearn. The criterion used for training was Gini. We experimented with various hyperparameters, such as 'max_depth', 'min_samples_split' etc, using GridSearch cross-validation. The model with the best performance was found to have hyperparameters: 'max_depth': 10, 'max_leaf_nodes': 40, 'min_samples_split': 4. The achieved accuracy on the train set was 97.25%. On the test set with a size of 346, the obtained accuracy was 97.3%. The train and test accuracies are nearly the same, indicating no overfit by the model at all.

E. Visualization and validation

Below are the evaluation metrics for the validation set: So

Classes	Precision	Recall	F_1 Score
Acc	0.94	0.97	0.96
Good	0.78	1.00	0.88
Unacc	1.00	0.98	0.99
Vgood	1.00	0.85	0.92

TABLE I
EVALUATION METRICS

we have fairly good F_1 scores for each class.

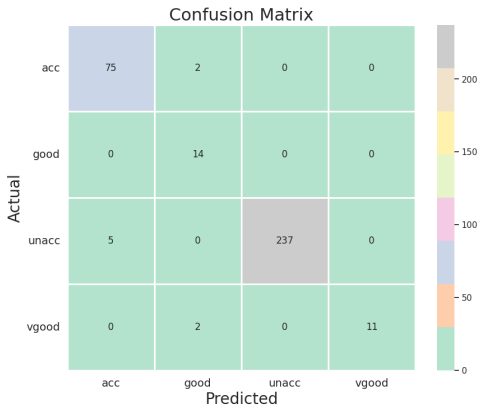


Fig. 6. Confusion Matrix with heatmap

In this confusion matrix for multiclass cases, all the diagonal numbers are the correct predictions made by the model. Non-diagonal terms are all incorrect predictions. The model with a lesser value of non-diagonal entries is the best one. The false positive, true positive, false negative and true negative notions are somewhat vague in the context of multiclass classification but still can be defined. From the confusion matrix, we get the accuracy as:

$$\text{Accuracy} = \frac{\text{Sum of diagonal terms}}{\text{Sum of all elements}}$$

$$\text{Accuracy} = \frac{75 + 14 + 237 + 11}{75 + 14 + 237 + 11 + 2 + 5 + 2} = 0.97$$

The goodness of a model can be tested by another metric called area under the ROC (Receiver operating characteristics) curve. The more the area, the better the model is. In the present scenario case, AU-ROC values for each of the classes are 0.99 for class Acc, 0.99 for class Good, 0.99 For class Unacc and 0.99 for class Vgood. Please note that Acc denotes Acceptable, Unacc implies Unacceptable, and Vgood is Very good. Earlier, the AUC for vgood class was 0.96. Now it is improved. So, with all this analysis, the model performance is undoubtedly extremely good.

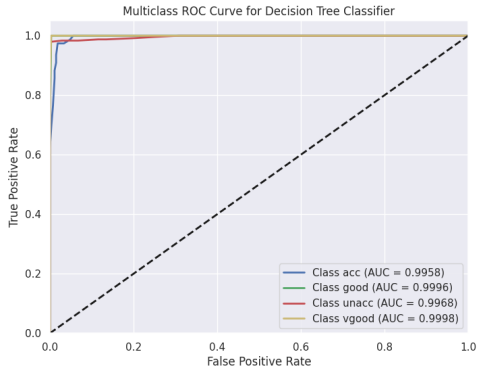


Fig. 7. Receiver operating characteristic curve

1) Insights in test data and visual validation of model: Bar plots are analysed in the test set to gain insights from the data. In addition to that, this visualization will help assess the model performance manually.

- The following plot is the acceptance of a car based on the purchase price. This plot is similar to the one of the exploratory data analysis. The number of unacceptable cars is the highest in all the price ranges. The acceptance of cars is the least in the case of the most expensive ones. This is quite obvious. This trend is illustrated in the same way in both the original labelled data and the predicted data.

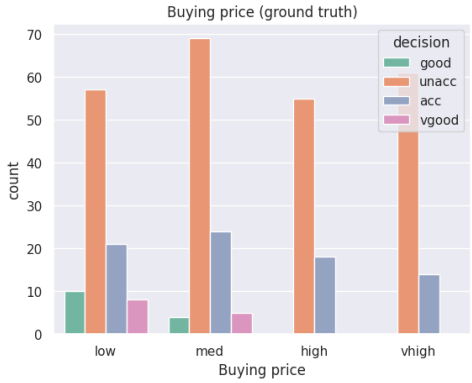


Fig. 8. Acceptabilty of the car based on buying price (original data)

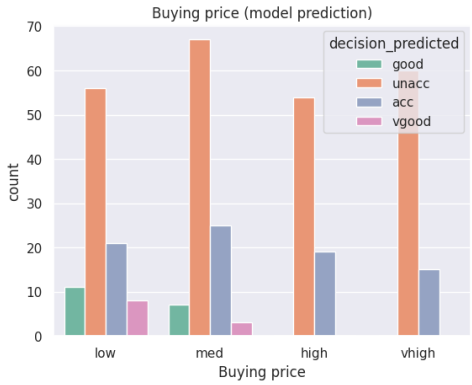


Fig. 9. Acceptabilty of the car based on buying price (model prediction)

- The maintenance plot below shows that cars with higher maintenance costs have less acceptability; the same holds for the prediction graph. Obviously, common people won't prefer cars with much higher maintenance costs.

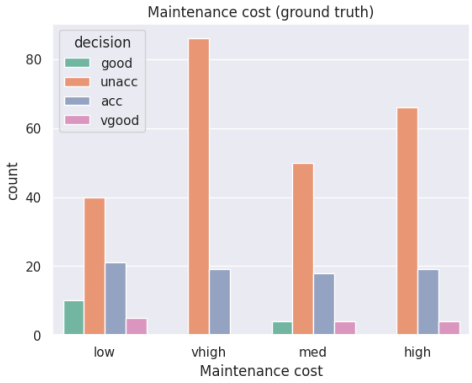


Fig. 10. Acceptance based on cost of maintenance (original data)

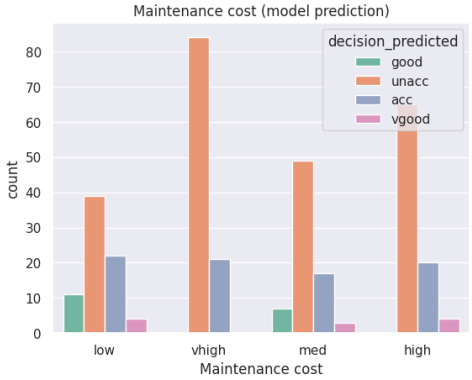


Fig. 11. Acceptance based on cost of maintenance (predicted data)

Fig. 12. Acceptance based on passenger capacity (original data)

- From the two figures above and below, it is seen that cars with only 2 passenger capacity are unacceptable. Our general notion says these kinds of cars are primarily race cars, which are really expensive, and for common people with families, a two-seater car is completely useless. We see the largest number of acceptable cars are four-seaters. The trend is the same in both the prediction and original plots.

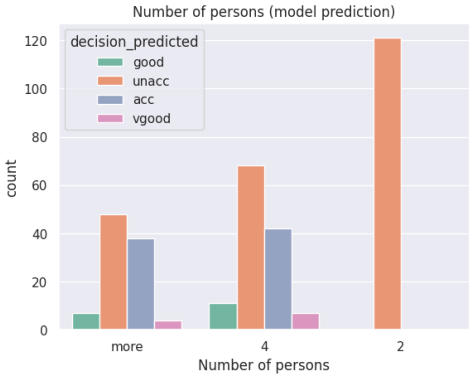


Fig. 13. Acceptance based on passenger capacity (model prediction)

- The plot given below provides information regarding the relation between a car's safety and acceptance. Like the one in the EDA part, a highly unsafe car is unacceptable. This is quite reasonable because no one will prefer to purchase a car whose safety is very low.

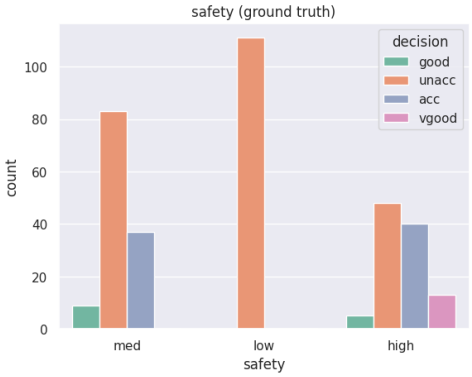


Fig. 14. Safety and acceptance in original data

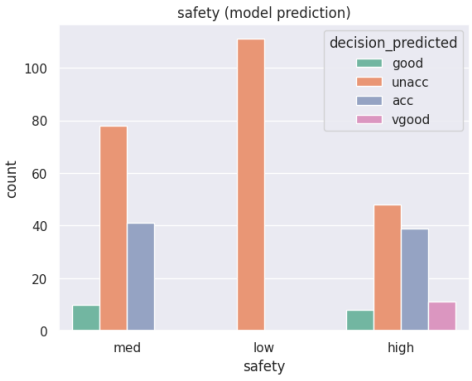


Fig. 15. Safety and acceptance in the model prediction data

- Further, it is seen that the number of rejected cars is higher for moderately safe cars, too. The acceptability is highest for the safest cars. And the difference between the number of unacceptable and acceptable cars, which are the safest, is also very small. So evidently, people prefer

the safest cars over anything else. The true and predicted labels have quite a high similarity.

So, this visualization really helped to understand the model interpretability a great deal. The similarity between the original and the predicted classes proves that the model is very good in terms of reliability.

II. Conclusions

Decision tree is one of the most popular classification algorithms. It is quite easy to explain when the tree depth is less. Unlike Logistic regression, it can work on both categorical and numerical data. It has the ability to capture complex patterns in the data, which many classification algorithms like Logistic regression, Naive Bayes, and Support vector machines often fail. With the grid search cross-validation, our max depth has been reduced to only 10, which is much easier for model explainability. The model performance is almost the same as the one in assignment 4. The gain is the tree depth and reduction in the complexity of the tree. A cross-validated model is much more robust and stable against overfitting.

References

[1] Decison tree: https://www.geeksforgeeks.org/decision-tree/
[2] Evaluation Metrics: https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/
[3] Pandas: https://pandas.pydata.org/
[4] Scikit learn Decision tree: https://scikit-learn.org/stable/modules/tree.html
[5] Seaborn: https://seaborn.pydata.org/index.html
[6] GridsearchCV: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Random Forest

Anik Bhowmick
Inter Disciplinary Dual Degree Data-Science
Indian Institute of Technology Madras
ae20b102@smail.iitm.ac.in

***Abstract*—This assignment discusses the application of a famous ensemble technique called Random Forest on the Car Evaluation Data Set. This data was derived from a simple hierarchical decision model originally developed for the demonstration of DEX, M. Bohanec, V. Rajkovic: Expert system for decision making. Sistemica 1(1), pp. 145-157, 1990.). The data comprises various kinds of car features, such as the buying price, maintenance price, maximum passenger capacity, safety, etc. Based on it, the task is to predict whether a car is acceptable or not, depending on its safety. The Problems section will be highlighted for the model performance improvement in this revised version of the assignment.**

***Index Terms*—Visualization, Ensemble Methods, Bagging, Random Forest, One hot encoding, Cramer's V rule, Correlation coefficient, Confusion Matrix, Accuracy, Precision, Recall, F_1 score, ROC.**

I. The Problem

As mentioned, our goal in this assignment is to predict whether a car is safe, acceptable, or unacceptable.

A. Cleaning and preparing the data

1) Encoding the categorical columns: Because any ML model can't work with strings, we need numerical representations of the categorical features. There are two kinds of encoding primarily used in Machine learning tasks. One hot encoding and Label encoding. This time, we decided to go for One hot encoding to see how the model is behaving to sparse data points. The one hot encoding is implemented in the form of a Pipeline so that we don't tamper with the original data. The data will be modified during the training time, and that modified data will fed to the model. The advantage is that the original data is preserved without creating any extra duplicates of it.

B. Exploratory analysis

1) Correlation among the numerical featurers: Two features are correlated if they have an absolute correlation coefficient close to 1. They are uncorrelated if their correlation coefficient is closer to 0. For categorical features, the conventional person correlation technique can not be applied. There is a technique called Cramer's V correlation is widely used for this purpose. It ranges from 0 to 1, indicating no association to a perfect association, respectively. First, the contingency table is calculated. It lists frequencies or counts of the combinations of every two categorical variables. Then the chi-squared test is performed.

$$\chi^2 = \sum \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

Here O_{ij} is the observed frequency in each cell, and E_{ij} is the expected frequency in each cell. The Cramer's V rule is given as :

$$V = \sqrt{\frac{\chi^2}{n \times \min(k-1, r-1)}}$$

Where n is the total number of observations, k is the number of categories in a variable, and r is the number of categories in another variable.

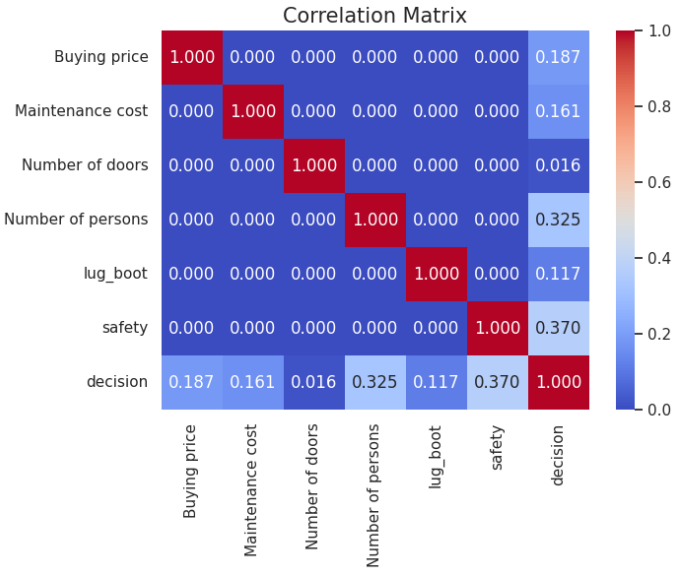


Fig. 1. Correlation plot

All the input categorical features are uncorrelated because the values of correlation coefficients are zero. So, there is no need to drop any column.

2) Preliminary study on the trend followed by data: We will check the class distribution to get an idea if the data is imbalance or not.

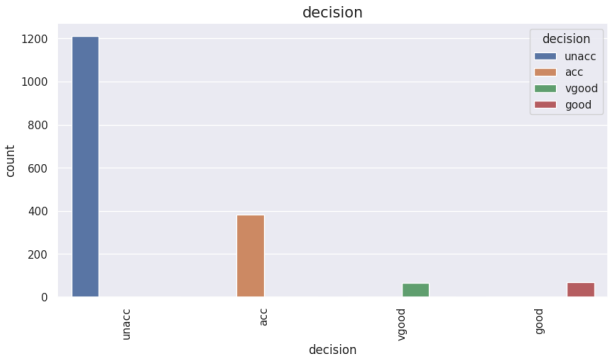


Fig. 2. Absolute count of 4 classes

This plot clearly conveys the fact that classes are highly imbalanced, with the unacceptable class being the largest in number. This gives a clear indication that accuracy can not be used for the model validation. Other metrics, such as precision, recall and F_1 score, need to be used. The true counts of these 4 classes are unacc: 1210, acc: 384, good: 69, vgood: 65.

C. Splitting the Training data to Train and validation sets

The data will split into training and test sets to validate the model, with the test size being 30%. Because of the imbalanced dataset, it is highly necessary to make stratified splits. Because it is never desirable that the training set does not contain any minority classes viz, good and vgood at all, and all the good, vgood classes are only present in the test set. Naturally, the model outputs will be full of errors, and the model might not even see the minority classes during training. So, stratification is very important for this dataset. This ensures both the train and test set contain all the classes with almost similar ratios.

D. Statistical model

Our model is a Random forest classifier from sklearn. In order to perform proper hyperparameter tuning, the models are trained with the Grid search cross-validation technique. There were several parameters chosen, such as the number of estimators, the minimum number of leaf nodes, the maximum number of features, etc. With the best hyperparameters, the model achieved fairly high performance. It gave 98% accuracy on the holdout test set. The earlier model was giving 99% accuracy. But we should not look at it as a loss. In earlier times in assignment 5, the model was trained with ordinally encoded features with a test size of 346, and now it is 518. Ordinal encoding sometimes introduces errors in the model. It is because we know the fact that the data is ordinally encoded. But to the model, these are just some integers. Now, it may so happen that the model is giving extra attention to the larger integers than the smaller ones. This is completely meaningless in the case of categorical data. We can compromise with 98% accuracy, considering the current model treats every data point with equal weightage and works with larger test data. Cross-validation ensures the model is free from overfitting

E. Visualization and validation

Below are the evaluation metrics for the validation set: The

Classes	Precision	Recall	F_1 Score
Acc	0.96	0.95	0.95
Good	0.95	1.00	0.98
Unacc	0.99	0.99	0.99
Vgood	0.95	1.00	0.98

TABLE I
EVALUATION METRICS

recall of minority-class Vgood has increased quite a bit. Now it is 1.00. Earlier it was 0.92.

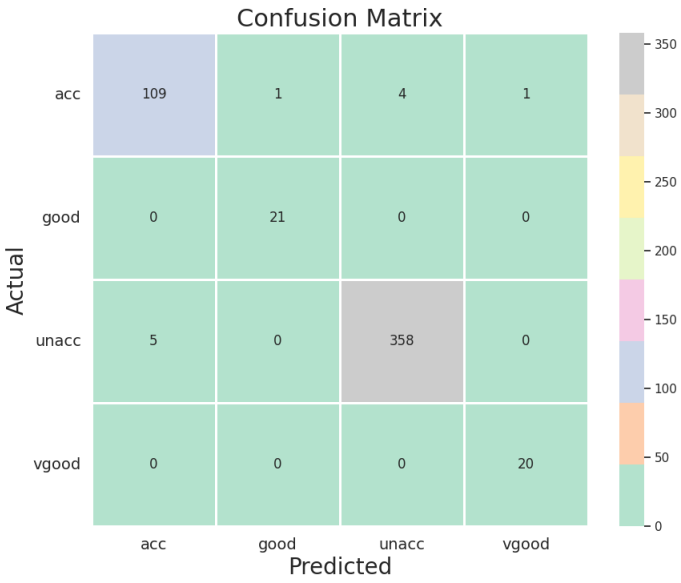


Fig. 3. Confusion Matrix with heatmap

In this confusion matrix for multiclass case, all the diagonal numbers are the correct predictions made by the model. Non-diagonal terms are all incorrect predictions. The model with a lesser value of non-diagonal entries is the best one. The false positive, true positive, false negative and true negative notions are somewhat vague in the context of multiclass classification but still can be defined. From the confusion matrix, we get the accuracy as:

$$\text{Accuracy} = \frac{\text{Sum of diagonal terms}}{\text{Sum of all elements}}$$

$$\text{Accuracy} = \frac{109 + 21 + 358 + 20}{109 + 21 + 358 + 20 + 1 + 4 + 1 + 5} = 0.98$$

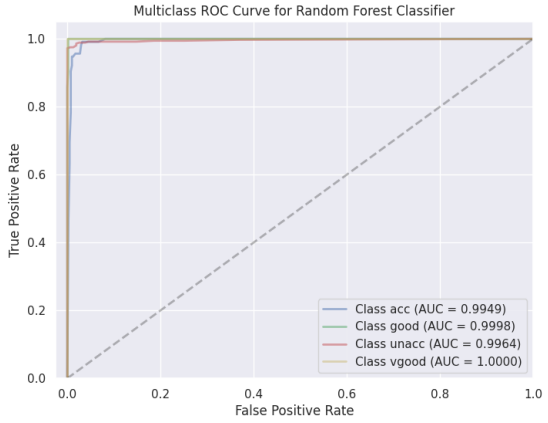


Fig. 4. Receiver operating characteristic curve

The goodness of a model can be tested by another metric called area under the ROC (Receiver operating characteristics) curve. The more the area, the better the model is. In the present scenario case, AU-ROC values for each of the classes are 0.99 for class Acc, 0.99 for class Good, 0.99 For class Unacc and 1.00 for class Vgood. Please note that Acc denotes Acceptable, Unacc implies Unacceptable, and Vgood is Very good. So, with all this analysis, the model performance is undoubtedly extremely good.

1) Insights in test data and visual validation of model: Bar plots are analysed in the test set to gain insights from the data. In addition to that, this visualization will help assess the model performance manually.

- The following plot is the acceptance of a car based on the purchase price. This plot is similar to the one of the exploratory data analysis. The number of unacceptable cars is the highest in all the price ranges. The acceptance of cars is the least in the case of the most expensive ones. This is quite obvious. This trend is illustrated in the same way in both the original labelled data and the predicted data.

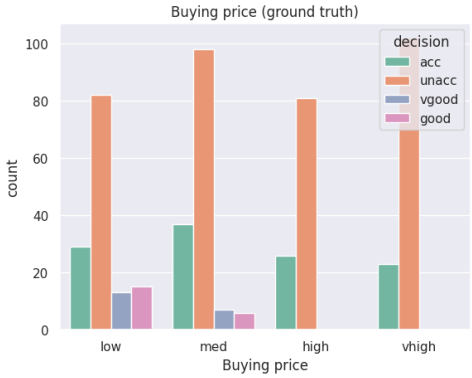


Fig. 5. Acceptabilty of the car based on buying price (original data)

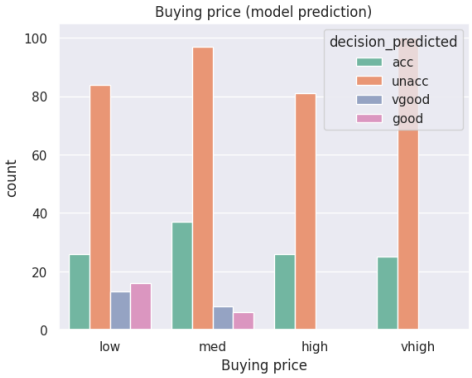


Fig. 6. Acceptabilty of the car based on buying price (model prediction)

- The cars that are costly to maintain are usually less acceptable. The bar plots below also convey the same message.

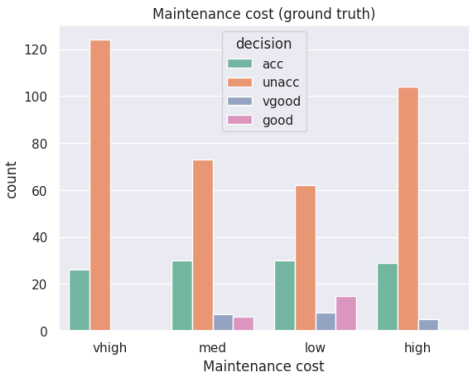


Fig. 7. Acceptance based on passenger capacity (original data)

The bar plot of predicted data is quite similar to the above plot. In both cases, there is no entry of the vgood and good class for the cars that charge the highest for maintenance. This implies people don't consider these kind of cars good or very good in terms of safety.

Fig. 8. Acceptance based on passenger capacity (model prediction)

The cars that have very low maintenance costs have the least unacceptance frequency, which is quite relevant.

The number of acceptable cars across all the types of low, high, high, and med is more or less the same. The class with the lowest frequency is vgood across all the price ranges of car maintenance. Both ground truth and predicted ones are exactly identical in this case.

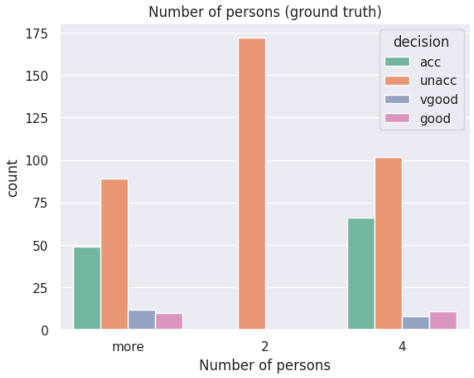


Fig. 9. Acceptance based on passenger capacity (original data)

- From the two figures above and below, it is seen that cars with only 2 passenger capacity are unacceptable. Our general notion says these kinds of cars are primarily race cars, which are really expensive, and for common people with families, a two-seater car is completely useless. We see the largest number of acceptable cars are four-seaters. The trend is the same in both the prediction and original plots.

Fig. 10. Acceptance based on passenger capacity (model prediction)

- The plot given below provides information regarding the relation between a car's safety and acceptance. Like the one in the EDA part, a highly unsafe car is unacceptable. This is quite reasonable because no one will prefer to purchase a car whose safety is very low.

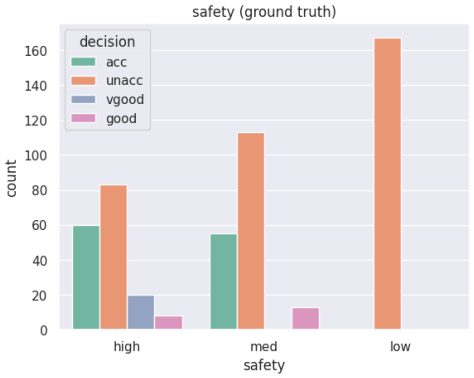


Fig. 11. Safety and acceptance in original data

Further, it is seen that the number of rejected cars is higher for moderately safe cars, too. The acceptability is highest for the safest cars. And the difference between the number of unacceptable and acceptable cars, which are the safest, is also very small. So evidently, people prefer the safest cars over anything else. The true and predicted labels have quite high similarity.

Fig. 12. Safety and acceptance in the model prediction data

So, this visualization really helped to understand the model interpretability a great deal. The similarity between the original and the predicted classes proves that the model is very good in terms of reliability.

II. Conclusions

Random Forest is one of the most popular ensemble algorithms. Unlike Logistic regression, it can work on both categorical and numerical data. It has the ability to capture complex patterns in the data, which many classification algorithms like Logistic regression, Naive Bayes, and Support vector machines often fail. The classification capacity of this algorithm is way higher than that of conventional decision trees just because of the fact that it uses multiple decision trees. The use of multiple decision trees makes it very robust against the variance of the data. In this revisited Random forest assignment, we have successfully implemented the model with proper hyperparameters for larger test data. The model didn't

lose its capacity significantly but proved its robustness against high-variance data.

REFERENCES

[1] Random Forest: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

[2] Evaluation Metrics: https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/

[3] Pandas: https://pandas.pydata.org/

[4] Scikit learn gridsearchcv: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[5] Scikit learn random forest: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[6] Seaborn: https://seaborn.pydata.org/index.html

Support Vector Machine

Anik Bhowmick
Inter Disciplinary Dual Degree Data-Science
Indian Institute of Technology Madras
ae20b102@smail.iitm.ac.in

***Abstract*—This assignment discusses the application of the Support Vector Machine on the Pulsar Data Set. HTRU2 actually created this data. The data comprises various kinds of star features, such as the Mean of the integrated profile, Standard deviation of the integrated profile, Excess kurtosis of the integrated profile, etc. Based on it, the task is to predict whether a star is a pulsar star or not. Pulsars are a rare type of Neutron star. The electromagnetic spectrum they produce is detectable from the earth. Currently, various machine learning techniques are employed to automate the detection of this kind of stars. These stars are very important for current scientific research, the study of interstellar mediums, and many more. This revised version of the SVM assignment aims to improve the model performance than before.**

***Index Terms*—Visualization, SVM classifier, Bagging classifier, Correlation coefficient, Confusion Matrix, SMOTE, Accuracy, Precision, Recall, F_1 score, ROC.**

I. The Problem

As mentioned, our goal in this assignment is to predict whether a neutron star is pulsar or not.

A. Cleaning and preparing the data

This data is completely numerical, with some missing entries.

1) Missing Value imputation: The general approach for imputing missing values of continuous variables is either mean or median.

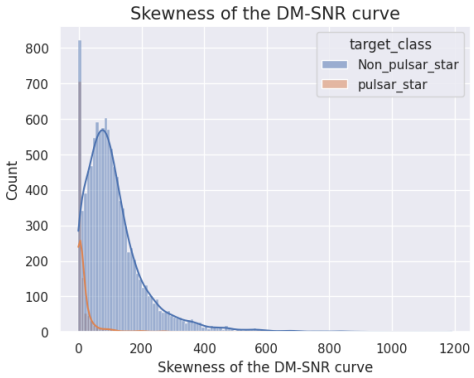


Fig. 1. Skewness of DM-SNR curve

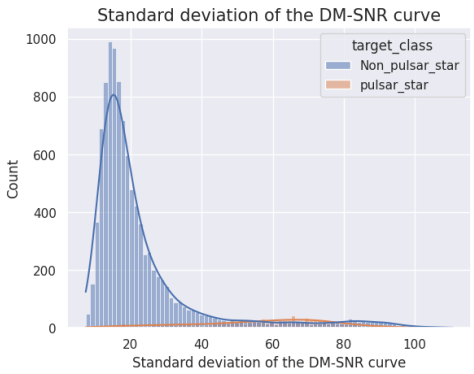


Fig. 2. Standard deviation of DM-SNR curve

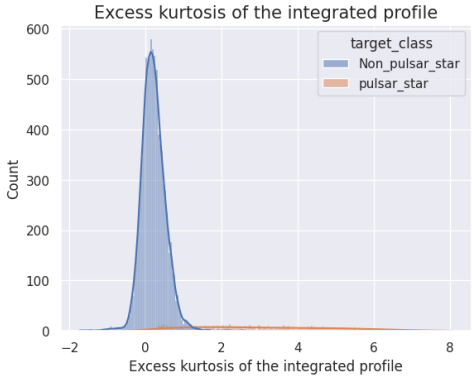


Fig. 3. Excess kurtosis of integrated profile

Mean is a good choice when the distribution is close to normal. However, the median is preferred for replacing the missing values for a skewed dataset. The three distribution plots for three features containing missing values given above are all skewed. So, median imputation is the best for each. But we used a different imputation technique, called K nearest neighbour imputation. KNN uses neighbouring data points to decide what value to put at the missing place. This is quite useful for numerical data. K-Nearest Neighbors (KNN) imputation is a method for filling in missing values in a dataset. It works by identifying the 'k' nearest neighbours to each missing value using a similarity metric. The missing value is then calculated based on the values of these neighbours, often using an average or weighted average approach. By leveraging

the information from nearby data points, KNN imputatio provides a way to estimate missing values and improve th completeness of the dataset.

B. Exploratory analysis

1) Correlation among the numerical featurers: Two fea tures are correlated if they have an absolute correlation co efficient close to 1. They are uncorrelated if their correlatio coefficient is closer to 0. The Pearson's correlation coefficien is given as follows:

$$r_{xy} = \frac{\sum_i x_i y_i - n\bar{x}\bar{y}}{\sqrt{\sum_i x_i^2 - n\bar{x}^2}\ \sqrt{\sum_i y_i^2 - n\bar{y}^2}}.$$

Fig. 5. Train set correlation heatmap after dropping features

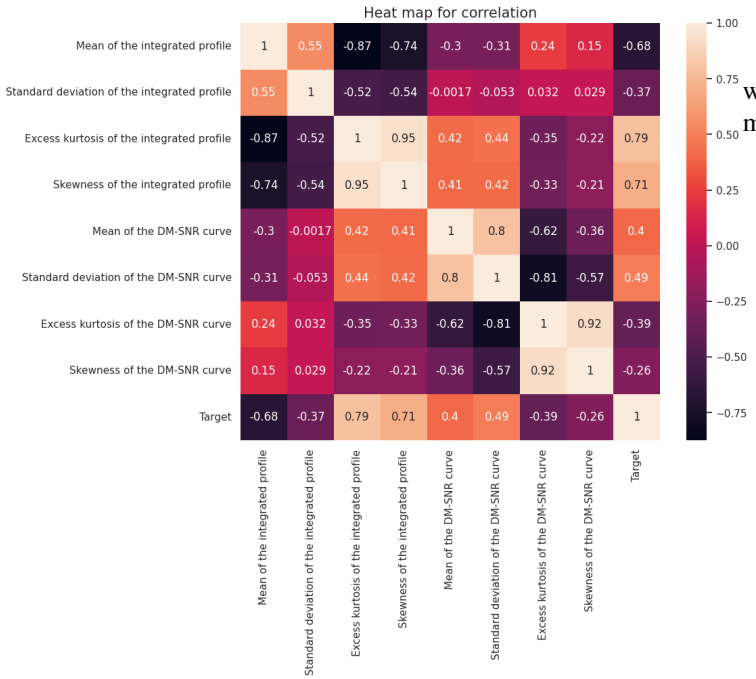


Fig. 4. Train set correlation heatmap before dropping features

2) Preliminary study on the trend followed by data: We will study a few plots before finally feeding the data to the model.

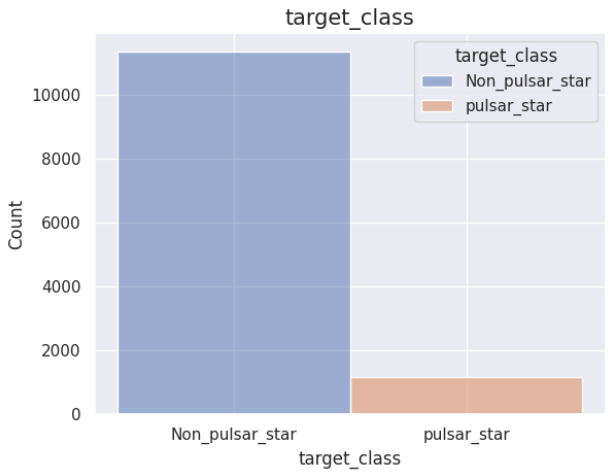


Fig. 6. Target class frequency of both classes

The features 'skewness of integrated profile' and 'skewness of DM SNR curve' are correlated to some other features, as seen in the plot above, having a correlation coefficient greater than 0.9. So, we will drop these two features. Correlated features lead to erroneous model training. After dropping these two features, the correlation plot looks as below:

This plot shows that the dataset is highly imbalanced, with 11375 instances of non-pulsar stars and 1153 instances of pulsar stars. We will handle this issue with a technique called SMOTE (Synthetic Minority Oversampling Technique).
We will look into some distribution plots before finally feeding the data into the model.

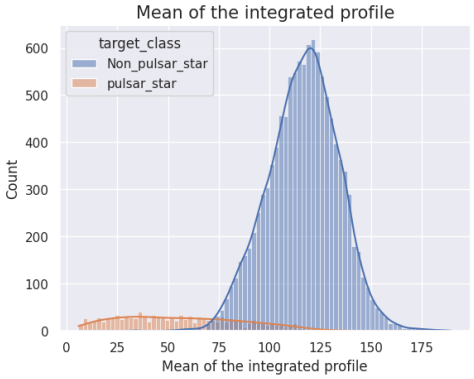


Fig. 7. Mean of the integrated profile in the train set

This is almost normally distributed from the majority class.

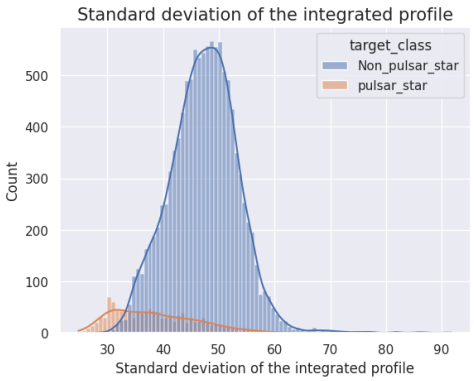


Fig. 8. Standard deviation of integrated profile in the train set

This distribution is a little left-skewed for both classes.

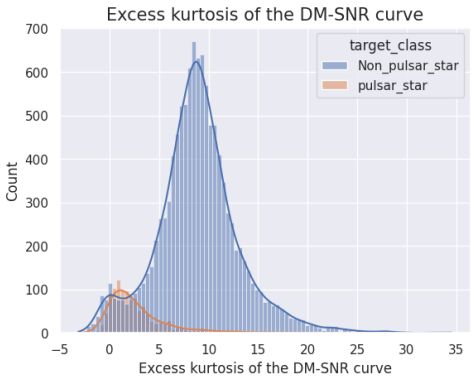


Fig. 9. Excess kurtosis of DM-SNR curve in the train set

This preliminary distribution visualization is necessary because we will crosscheck if this same distribution is followed by the test set later.

C. Splitting the Training data to Train and validation sets

The data set given to us is already split into train and test sets, with around 5000 entries in the test set. So, we need not perform a train test split again.

D. Statistical model

The earlier model was a single Support vector machine classifier from sklearn. This time, as a part of ensemble learning, we decided to use 100 highly overfit SVM classifiers. Each has C= 0.1 and a 10-degree polynomial kernel. Bootstrapping for both examples and features was performed to improve the model. To eliminate the issue of data scarcity in minor classes, SMOTE is adopted. SMOTE is an oversampling technique that generates synthetic samples for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling. It focuses on the feature space to generate new instances with the help of interpolation. Everything was implemented inside a pipeline so that original data was not tampered with by any means. The SMOTE resulted in the generation of a balanced data set where the number of datapoints in both classes are equal.

E. Visualization and validation

Below are the evaluation metrics for the train set:

Classes	Precision	Recall	F_1 Score
Non pulsar star	0.98	0.99	0.99
Pulsar star	0.85	0.91	0.88

TABLE I
EVALUATION METRICS

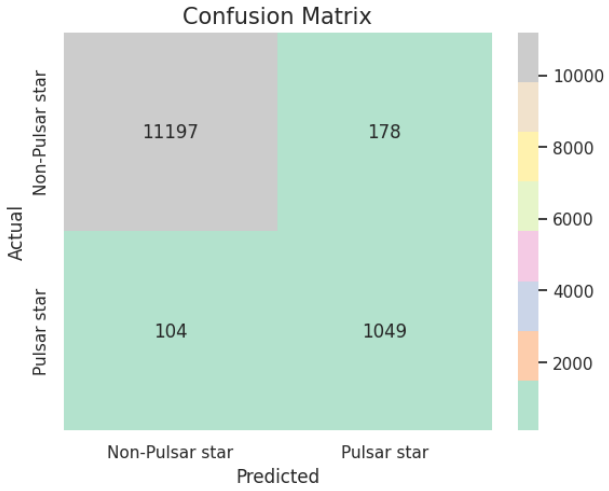


Fig. 10. Confusion Matrix with heatmap

From this figure, we can infer the following

- **True Positive**=1049 (Actual Pulsar star, predicted Pulsar star)
- **False Positive**=178 (Actual Non-pulsar star, predicted pulsar star)
- **False Negetive**=104 (Actual Pulsar star, predicted Non-pulsar star)
- **True Negative**=11197 (Actual Non-pulsar star, predicted Non-pulsar star)

From the confusion matrix, we get the accuracy as:

$$\text{Accuracy} = \frac{\text{Sum of diagonal terms}}{\text{Sum of all elements}}$$

$$\text{Accuracy} = \frac{11197 + 1049}{11197 + 1049 + 104 + 178} = 0.98$$

The earlier accuracy was 97%. This time, it is 98%. So, a slight improvement. The goodness of a model can be tested by another metric called area under the ROC (Receiver operating characteristics) curve. The more the area, the better the model is. In our case, the achieved AU-ROC is 0.98 (0.97 earlier), which is a fairly good indication of improvement.

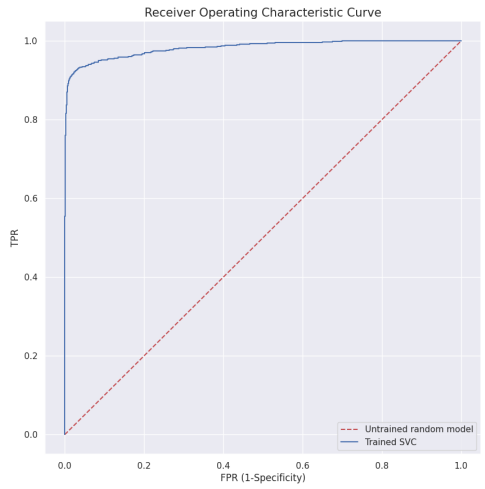


Fig. 11. Receiver operating characteristic curve

1) Insights in test data and visual validation of model:

- The relative class frequency (as predicted by the model) matches that of the train set. The Pulsar star class is also a minority class here too. The majority class has 4840, and the minority class has 530 entries.

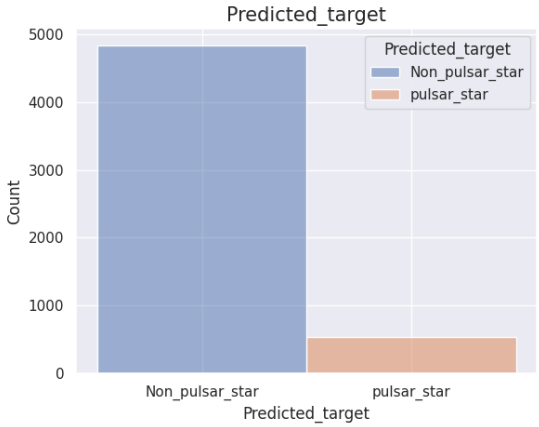


Fig. 12. Class count as predicted by the model in the test set

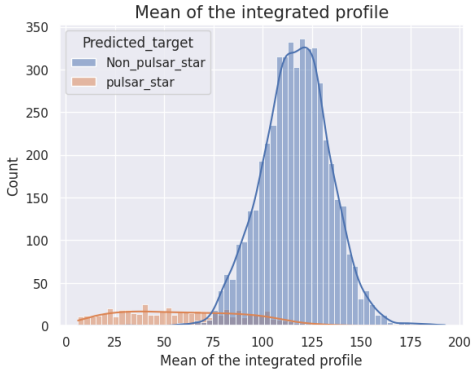


Fig. 13. Test set mean of integrated profile

- The test set mean integrated profile looks similar to the train set one. Even the labels look similar. So, the model follows the same trend as what it has seen in the train set.

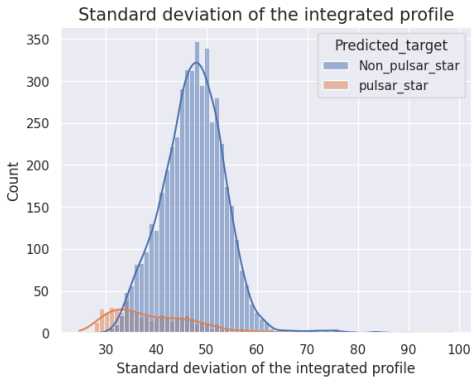


Fig. 14. Test set standard deviation of the integrated profile

- This distribution is skewed like the train set one in terms of both predicted classes.
- Excess kurtosis plot, as given below, is also quite similar to the train set.

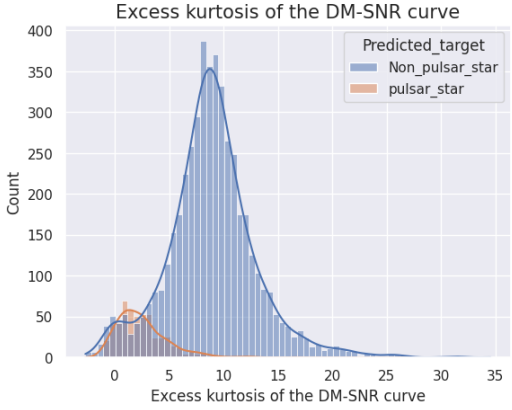


Fig. 15. DM-SNR curve excess kurtosis from the test set

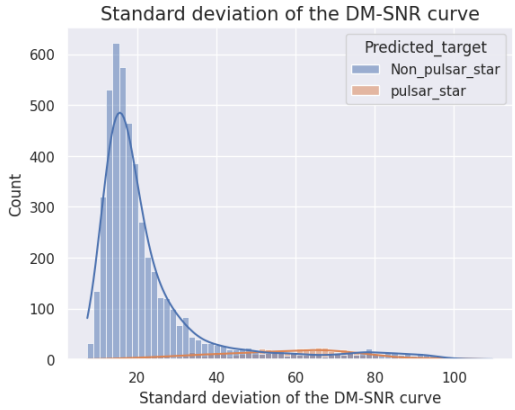


Fig. 16. Standard deviation of DM-SNR curve in the test set

So, all the distributions of the test set are similar to that of the train set. And the model behaves exactly the same way as it should.

II. Conclusions

Support vector machine is one of the most popular classification algorithms. Unlike other classification models, this model is very robust to the outliers, as it only considers a very small number of points to come up with a decision boundary. The classification capacity of this algorithm increases when the regularization parameter C is chosen properly along with the right kernel function. The primary motivation for SVM came from a very simple perceptron learning algorithm. In this assignment, we successfully demonstrated the applicability of the SVM classifier under the ensemble technique. This allowed us to get a generalized single SVM classifier from 100 high-variance SVM classifiers. The use of feature bagging and bootstrapping of examples undoubtedly made the resulting model very strong and free from high variance. In addition, we used SMOTE, which balances the major and minor classes by artificially increasing the number of training samples in the minority class. This helped in learning a strong bias-free model, too.

References

[1] Support vector machine: https://www.geeksforgeeks.org/support-vector-machine-algorithm/
[2] Evaluation Metrics: https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/
[3] Pandas: https://pandas.pydata.org/
[4] Scikit learn Support Vector Machines: https://scikit-learn.org/stable/modules/svm.html
[5] Imblearn SMOTE: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html
[6] SMOTE: https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/
[7] Seaborn: https://seaborn.pydata.org/index.html

DAL assignment Linear Regression (revised)

Importing all necessary libraries

In [ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler
!pip install eli5
import eli5
import numpy as np
from eli5.sklearn import PermutationImportance
from scipy import stats
import copy
```

```
Requirement already satisfied: eli5 in /usr/local/lib/python3.10/dist-packages (0.
13.0)
Requirement already satisfied: attrs>17.1.0 in /usr/local/lib/python3.10/dist-pack
ages (from eli5) (23.1.0)
Requirement already satisfied: jinja2>=3.0.0 in /usr/local/lib/python3.10/dist-pac
kages (from eli5) (3.1.2)
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.10/dist-pack
ages (from eli5) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (f
rom eli5) (1.11.3)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (fro
m eli5) (1.16.0)
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.10/dis
t-packages (from eli5) (1.2.2)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages
(from eli5) (0.20.1)
Requirement already satisfied: tabulate>=0.7.7 in /usr/local/lib/python3.10/dist-p
ackages (from eli5) (0.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-p
ackages (from jinja2>=3.0.0->eli5) (2.1.3)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-pac
kages (from scikit-learn>=0.20->eli5) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/d
ist-packages (from scikit-learn>=0.20->eli5) (3.2.0)
```

Importing the datasets from google drive

In [ ]:
```
merged_data=pd.read_excel('/content/drive/MyDrive/DAL dataset/merged_data.xlsx')
```

Obtaining the columns and information for the dataset

In [ ]:
```
merged_data.columns
```

Out[ ]:
```
Index(['Unnamed: 0', 'State', 'AreaName', 'All_Poverty', 'M_Poverty',
       'F_Poverty', 'FIPS', 'Med_Income', 'Med_Income_White',
       'Med_Income_Black', 'Med_Income_Nat_Am', 'Med_Income_Asian', 'Hispanic',
       'M_With', 'M_Without', 'F_With', 'F_Without', 'All_With', 'All_Without',
       'fips_x', 'Incidence_Rate', 'Avg_Ann_Incidence', 'recent_trend',
       'fips_y', 'Mortality_Rate', 'Avg_Ann_Deaths'],
      dtype='object')
```

In [ ]:
```
merged_data.rename(columns=lambda x: x.replace('_', ' '), inplace=True)
```

In [ ]:
```
merged_data.columns
```

Out[ ]:
```
Index(['Unnamed: 0', 'State', 'AreaName', 'All Poverty', 'M Poverty',
       'F Poverty', 'FIPS', 'Med Income', 'Med Income White',
       'Med Income Black', 'Med Income Nat Am', 'Med Income Asian', 'Hispanic',
       'M With', 'M Without', 'F With', 'F Without', 'All With', 'All Without',
       'fips x', 'Incidence Rate', 'Avg Ann Incidence', 'recent trend',
       'fips y', 'Mortality Rate', 'Avg Ann Deaths'],
      dtype='object')
```

Overall information

In [ ]:
```
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3134 entries, 0 to 3133
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         3134 non-null   int64  
 1   State              3134 non-null   object 
 2   AreaName           3134 non-null   object 
 3   All Poverty        3134 non-null   int64  
 4   M Poverty          3134 non-null   int64  
 5   F Poverty          3134 non-null   int64  
 6   FIPS               3134 non-null   int64  
 7   Med Income         3133 non-null   float64
 8   Med Income White   3132 non-null   float64
 9   Med Income Black   1924 non-null   float64
 10  Med Income Nat Am  1474 non-null   float64
 11  Med Income Asian   1377 non-null   float64
 12  Hispanic           2453 non-null   float64
 13  M With             3134 non-null   int64  
 14  M Without          3134 non-null   int64  
 15  F With             3134 non-null   int64  
 16  F Without          3134 non-null   int64  
 17  All With           3134 non-null   int64  
 18  All Without        3134 non-null   int64  
 19  fips x             3134 non-null   int64  
 20  Incidence Rate     3134 non-null   object 
 21  Avg Ann Incidence  3134 non-null   object 
 22  recent trend       3134 non-null   object 
 23  fips y             3134 non-null   int64  
 24  Mortality Rate     3134 non-null   object 
 25  Avg Ann Deaths     3134 non-null   object 
dtypes: float64(6), int64(13), object(7)
memory usage: 636.7+ KB
```

Studying and preprocessing the merged_dataset

In [ ]:
```
merged_data
```

Out[ ]:

	Unnamed: 0	State	AreaName	All Poverty	M Poverty	F Poverty	FIPS	Med Income	Med Income White	Med Income Black
0	0	AK	Aleutians East Borough, Alaska	553	334	219	2013	61518.0	72639.0	31250.0
1	1	AK	Aleutians West Census Area, Alaska	499	273	226	2016	84306.0	97321.0	93750.0
2	2	AK	Anchorage Municipality, Alaska	23914	10698	13216	2020	78326.0	87235.0	50535.0
3	3	AK	Bethel Census Area, Alaska	4364	2199	2165	2050	51012.0	92647.0	73661.0
4	4	AK	Bristol Bay Borough, Alaska	69	33	36	2060	79750.0	88000.0	NaN
...	...	...	...	...	...	...	...	...	...	...
3129	3129	WY	Sweetwater County, Wyoming	5058	2177	2881	56037	69022.0	69333.0	23535.0
3130	3130	WY	Teton County, Wyoming	1638	1026	612	56039	75325.0	77651.0	NaN
3131	3131	WY	Uinta County, Wyoming	2845	1453	1392	56041	56569.0	56532.0	NaN
3132	3132	WY	Washakie County, Wyoming	1137	489	648	56043	47652.0	48110.0	NaN
3133	3133	WY	Weston County, Wyoming	958	354	604	56045	57738.0	57842.0	NaN

3134 rows × 26 columns

In [ ]:
```
# columns Med_Income, Med_Income_White, Med_Income_Black, Med_Income_Nat_Am, Med_In
merged_data['Med Income'].fillna(merged_data['Med Income'].median(),inplace=True)
merged_data['Med Income White'].fillna(merged_data['Med Income White'].median(),inp
merged_data['Med Income Black'].fillna(merged_data['Med Income Black'].median(),inp
merged_data['Med Income Nat Am'].fillna(merged_data['Med Income Nat Am'].median(),i
merged_data['Med Income Asian'].fillna(merged_data['Med Income Asian'].median(),inp
merged_data['Hispanic'].fillna(merged_data['Hispanic'].median(),inplace=True)
```

In [ ]:
```
merged_data.info()#Updated information
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3134 entries, 0 to 3133
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         3134 non-null   int64  
 1   State              3134 non-null   object 
 2   AreaName           3134 non-null   object 
 3   All Poverty        3134 non-null   int64  
 4   M Poverty          3134 non-null   int64  
 5   F Poverty          3134 non-null   int64  
 6   FIPS               3134 non-null   int64  
 7   Med Income         3134 non-null   float64
 8   Med Income White   3134 non-null   float64
 9   Med Income Black   3134 non-null   float64
 10  Med Income Nat Am  3134 non-null   float64
 11  Med Income Asian   3134 non-null   float64
 12  Hispanic           3134 non-null   float64
 13  M With             3134 non-null   int64  
 14  M Without          3134 non-null   int64  
 15  F With             3134 non-null   int64  
 16  F Without          3134 non-null   int64  
 17  All With           3134 non-null   int64  
 18  All Without        3134 non-null   int64  
 19  fips x             3134 non-null   int64  
 20  Incidence Rate     3134 non-null   object 
 21  Avg Ann Incidence  3134 non-null   object 
 22  recent trend       3134 non-null   object 
 23  fips y             3134 non-null   int64  
 24  Mortality Rate     3134 non-null   object 
 25  Avg Ann Deaths     3134 non-null   object 
dtypes: float64(6), int64(13), object(7)
memory usage: 636.7+ KB
```

After thoroughly going through the data we saw states 'MN' , 'KS', 'NV' have '_' and '__' for all incidence rate and Avg_Ann_Incidence values. so we will drop this row because no information is available

In [ ]:
```
# After thoroughly going through the data we saw states 'MN' , 'KS', 'NV' have '_'
KS_state=merged_data[merged_data['State']=='KS']
MN_state=merged_data[merged_data['State']=='MN']
NV_state=merged_data[merged_data['State']=='NV']
merged_data.drop(merged_data[merged_data['State']=='KS'].index,axis=0,inplace=True)
merged_data.drop(merged_data[merged_data['State']=='MN'].index,axis=0,inplace=True)
merged_data.drop(merged_data[merged_data['State']=='NV'].index,axis=0,inplace=True)
```

In [ ]:
```
#Replace all 3 or less values with 3 in Avg_Ann_incidence column
merged_data['Avg Ann Incidence'].replace('3 or fewer',3,inplace=True)
```

In [ ]:
```
#MI state contains # appended to the number end on Incidence_Rate column
merged_data[merged_data['State']=='MI']
```

Out[ ]:

	Unnamed: 0	State	AreaName	All Poverty	M Poverty	F Poverty	FIPS	Med Income	Med Income White	Med Income Black
1225	1225	MI	Alcona County, Michigan	1575	709	866	26001	38033.0	38313.0	30000.0
1226	1226	MI	Alger County, Michigan	1213	635	578	26003	39300.0	38853.0	30000.0
1227	1227	MI	Allegan County, Michigan	14185	6447	7738	26005	54264.0	54439.0	42047.0
1228	1228	MI	Alpena County, Michigan	4898	2193	2705	26007	38829.0	38966.0	29375.0
1229	1229	MI	Antrim County, Michigan	3316	1451	1865	26009	46845.0	47290.0	30000.0
...	...	...	...	...	...	...	...	...	...	...
1303	1303	MI	Tuscola County, Michigan	8317	3963	4354	26157	43768.0	43816.0	24688.0
1304	1304	MI	Van Buren County, Michigan	14029	6400	7629	26159	46008.0	48232.0	19101.0
1305	1305	MI	Washtenaw County, Michigan	51965	25329	26636	26161	61003.0	66530.0	35301.0
1306	1306	MI	Wayne County, Michigan	440449	200486	239963	26163	41210.0	54162.0	26888.0
1307	1307	MI	Wexford County, Michigan	5631	2846	2785	26165	41534.0	41456.0	16050.0

83 rows × 26 columns

In [ ]:
```
# Eliminate the # from the end of the number
merged_data['Incidence Rate'].replace({' #':''},regex=True,inplace=True)
```

In [ ]:
```
# we have been given * is <=16. so we will replace *s with mean of numbers <=16 on
merged_data['Incidence Rate']=pd.to_numeric(merged_data['Incidence Rate'],errors='c
merged_data['Incidence Rate'].fillna(merged_data[merged_data['Incidence Rate']<=16]
```

Its given that *in the Mortality Rate column indicates <=16. We will replace* in Avg_Ann_Deaths column with least value of Avg_Ann_Incidence. Because number of 3s in Avg_Ann_Incidence entries are almost same as of *s Avg_Ann_Deaths

In [ ]:
```
# Its given that * in the Mortality Rate column indicates <=16. We will replace * i
merged_data['Mortality Rate']=pd.to_numeric(merged_data['Mortality Rate'],errors='c
```

```
merged_data['Mortality Rate'].fillna(merged_data[merged_data['Mortality Rate']<=16]
merged_data['Avg Ann Deaths']=pd.to_numeric(merged_data['Avg Ann Deaths'],errors='c
```

In [ ]:
```
merged_data[merged_data['Avg Ann Incidence']==3][['Avg Ann Incidence','Avg Ann Deat
```

Out[ ]:

	Avg Ann Incidence	Avg Ann Deaths
0	3	NaN
1	3	NaN
4	3	NaN
5	3	NaN
6	3	NaN
...	...	...
2842	3	NaN
2913	3	4.0
2956	3	NaN
3124	3	NaN
3128	3	NaN

235 rows × 2 columns

In [ ]:
```
merged_data['Avg Ann Deaths'].fillna(3,inplace=True)#filling na values with 3
```

In [ ]:
```
merged_data['Mortality Rate']
```

Out[ ]:
```
0       13.05
1       13.05
2       47.30
3       58.30
4       13.05
        ...
3129    28.40
3130    29.10
3131    22.10
3132    38.20
3133    43.50
Name: Mortality Rate, Length: 2925, dtype: float64
```

In [ ]:
```
merged_data['Avg Ann Deaths']
```

Out[ ]:
```
0        3.0
1        3.0
2       96.0
3        5.0
4        3.0
        ...
3129     9.0
3130     5.0
3131     4.0
3132     5.0
3133     4.0
Name: Avg Ann Deaths, Length: 2925, dtype: float64
```

In [ ]:
```
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2925 entries, 0 to 3133
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         2925 non-null   int64  
 1   State              2925 non-null   object 
 2   AreaName           2925 non-null   object 
 3   All Poverty        2925 non-null   int64  
 4   M Poverty          2925 non-null   int64  
 5   F Poverty          2925 non-null   int64  
 6   FIPS               2925 non-null   int64  
 7   Med Income         2925 non-null   float64
 8   Med Income White   2925 non-null   float64
 9   Med Income Black   2925 non-null   float64
 10  Med Income Nat Am  2925 non-null   float64
 11  Med Income Asian   2925 non-null   float64
 12  Hispanic           2925 non-null   float64
 13  M With             2925 non-null   int64  
 14  M Without          2925 non-null   int64  
 15  F With             2925 non-null   int64  
 16  F Without          2925 non-null   int64  
 17  All With           2925 non-null   int64  
 18  All Without        2925 non-null   int64  
 19  fips x             2925 non-null   int64  
 20  Incidence Rate     2925 non-null   float64
 21  Avg Ann Incidence  2925 non-null   int64  
 22  recent trend       2925 non-null   object 
 23  fips y             2925 non-null   int64  
 24  Mortality Rate     2925 non-null   float64
 25  Avg Ann Deaths     2925 non-null   float64
dtypes: float64(9), int64(14), object(3)
memory usage: 617.0+ KB
```

'All_with' is sum of 'M_with' and 'F_with' columns. So male female individually needs not be considered. Same is true for poverty case and insurance case**

In [ ]:
```
merged_data[['M Without', 'F Without', 'All Without']]
```

Out[ ]:

	M Without	F Without	All Without
0	1317	540	1857
1	769	564	1333
2	23245	21393	44638
3	2708	1774	4482
4	124	67	191
...	...	...	...
3129	3318	2683	6001
3130	2558	1192	3750
3131	1413	1503	2916
3132	691	703	1394
3133	454	314	768

2925 rows × 3 columns

Dropping the non important features

In [ ]:
```
drop_features=['M Without','F Without','M With', 'F With','M Poverty', 'F Poverty',
merged_data.drop(drop_features,axis=1,inplace=True)
```

In [ ]:
```
merged_data.describe()
```

Out[ ]:

	All Poverty	Med Income	Med Income White	Med Income Black	Med Income Nat Am	Med Income Asian
count	2.925000e+03	2925.000000	2925.000000	2925.000000	2925.000000	2925.000000
mean	1.583774e+04	46549.887863	49339.476581	32958.151111	41037.877607	62869.864957
std	5.601920e+04	12404.089733	12649.915221	14603.092556	16572.327584	23002.278641
min	1.000000e+01	19328.000000	19340.000000	2499.000000	2499.000000	2499.000000
25%	1.884000e+03	38307.000000	41176.000000	26978.000000	39014.000000	60405.000000
50%	4.509000e+03	44589.000000	47074.000000	30000.000000	39014.000000	60405.000000
75%	1.112700e+04	52082.000000	54511.000000	32605.000000	39014.000000	60405.000000
max	1.800265e+06	123453.000000	136311.000000	170195.000000	250001.000000	250001.000000

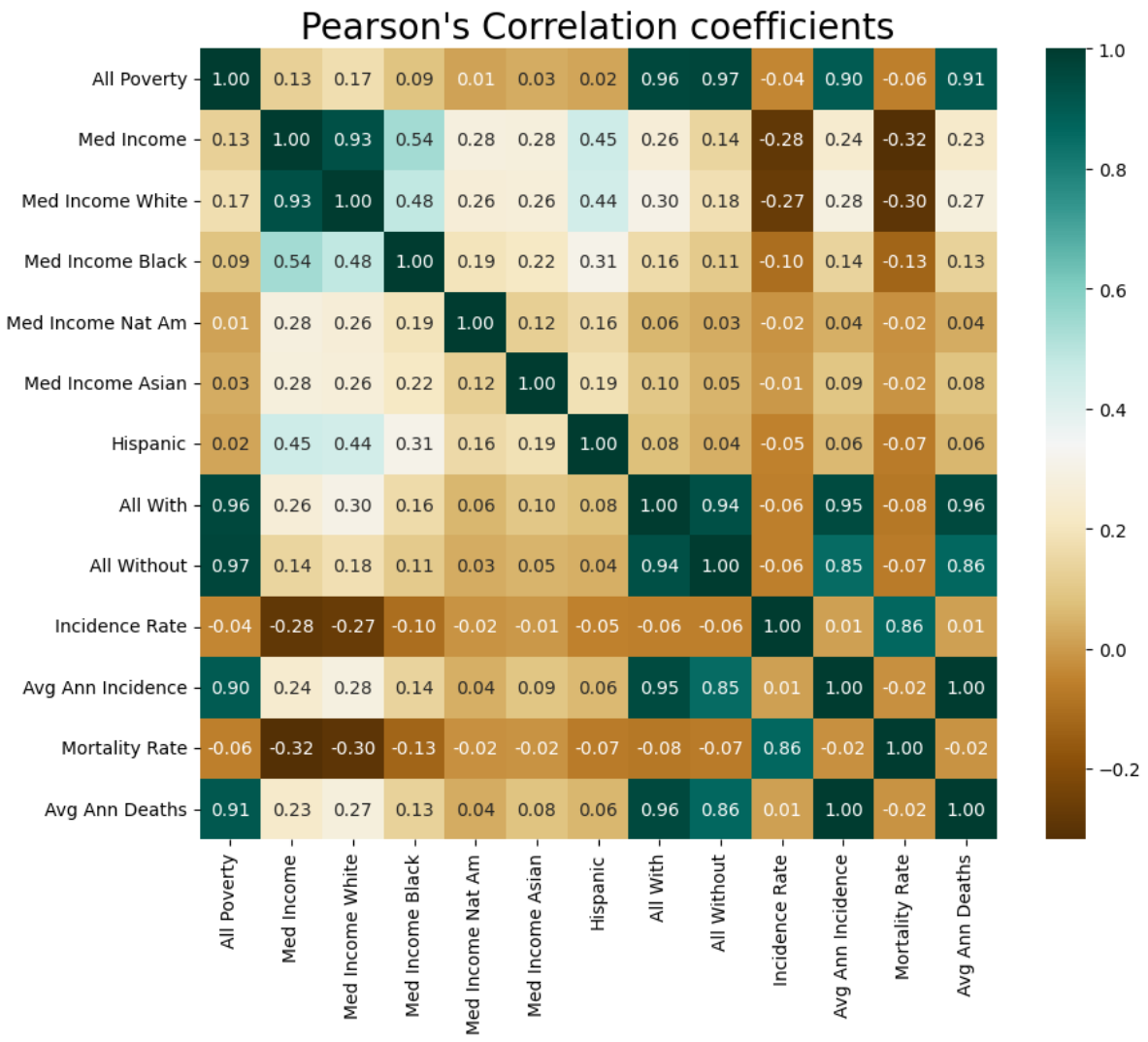
Exploratory Data Analysis

In [ ]:
```
def get_corr_plot(data):
  """
  gives the correlation heatmap
  data : a dataframe

  """
  plt.figure(figsize=(10,8))
  sns.heatmap(data.corr(),annot=True,cmap='BrBG',fmt='.2f')
  plt.title("Pearson's Correlation coefficients",fontsize=20)
  plt.show()
def get_box_plots(data,save=False):
  """
  gives the box plot and distribution for data to visually inspect the outliers
  data : dataframe
  save : saves the images when set to True

  """
  sns.set()
  for columns in data.columns:
    fig, axes = plt.subplots(2, 1,figsize=(20,20))
    fig.suptitle("Box and distribution plot for "+columns,fontsize=25)
    axes[0].set_title("Boxplot",fontsize=25)
    sns.boxplot(x=data[columns],ax=axes[0])
    axes[0].tick_params(axis='x', labelsize=20)
    axes[0].tick_params(axis='y', labelsize=20)
    axes[0].set_xlabel("Values",fontsize=20)
    axes[1].set_title("Distribution",fontsize=25)
    sns.kdeplot(data[columns],ax=axes[1])
    axes[1].set_ylabel("Density",fontsize=20)
    axes[1].tick_params(axis='x', labelsize=20)
    axes[1].tick_params(axis='y', labelsize=20)
```

```
    axes[1].set_xlabel("Values",fontsize=20)
    if save==True:
      fig.savefig(columns+".png")
    plt.show()
def get_residual_plot(y,yhat,col,save=False):
  """
  gives the residual plot along with distribution of the errors
  y : true value
  yhat : predicted value
  col : fetches the name of the output variable
  save : saves the image when set to True

  """
  resid=yhat-y
  fig, axes = plt.subplots(1, 2,gridspec_kw={'width_ratios': [10,8]})
  fig.suptitle("Residual Plot and Distribution for "+col)
  axes[0].scatter(yhat,resid,alpha=0.2)
  axes[0].set_xlabel("Predicted Target $\hat{y}$")
  axes[0].set_ylabel("Residuals")
  axes[0].axhline(y=0,color='black')
  axes[0].set_title("Residual Plot")
  axes[0].set_ylim(-60,60)
  sns.kdeplot(y=resid,ax=axes[1])
  axes[1].set_ylim(-60,60)
  axes[1].set_xlim(0.07,0)
  axes[1].set_title("Distribution")
  axes[1].set_ylabel(None)
  #axes[1].set_ylabel("Residuals")
  axes[1].set_xlabel("PDF of residual")
  if save==True:
    plt.savefig("Residual_density.png")
  plt.show()
```

Get the correlation among the features

In [ ]:
```
get_corr_plot(merged_data)
```



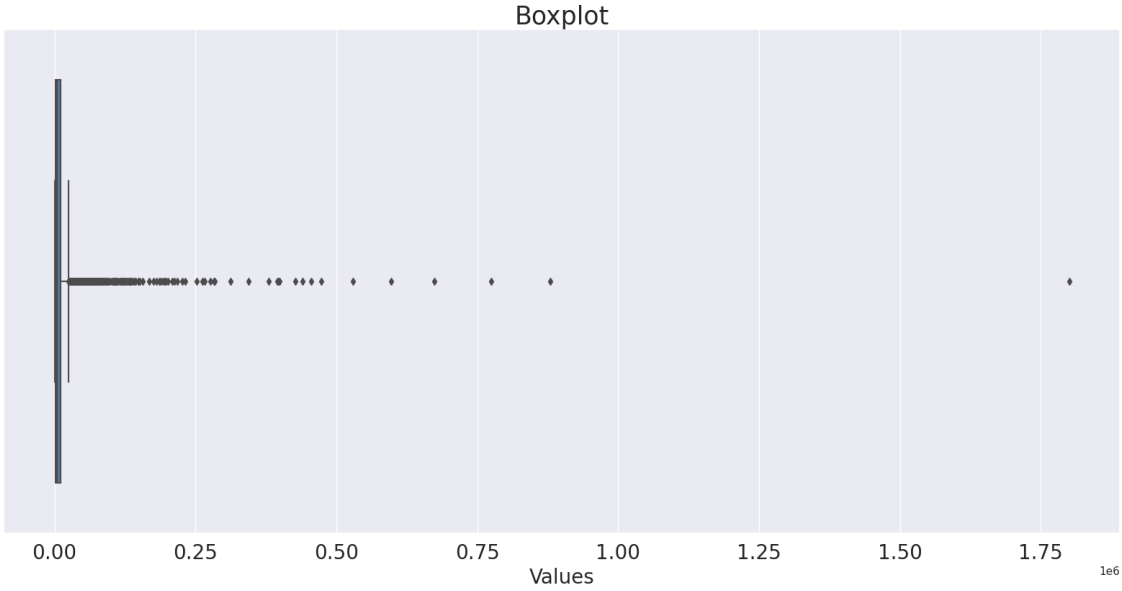
Making dataset for four different models

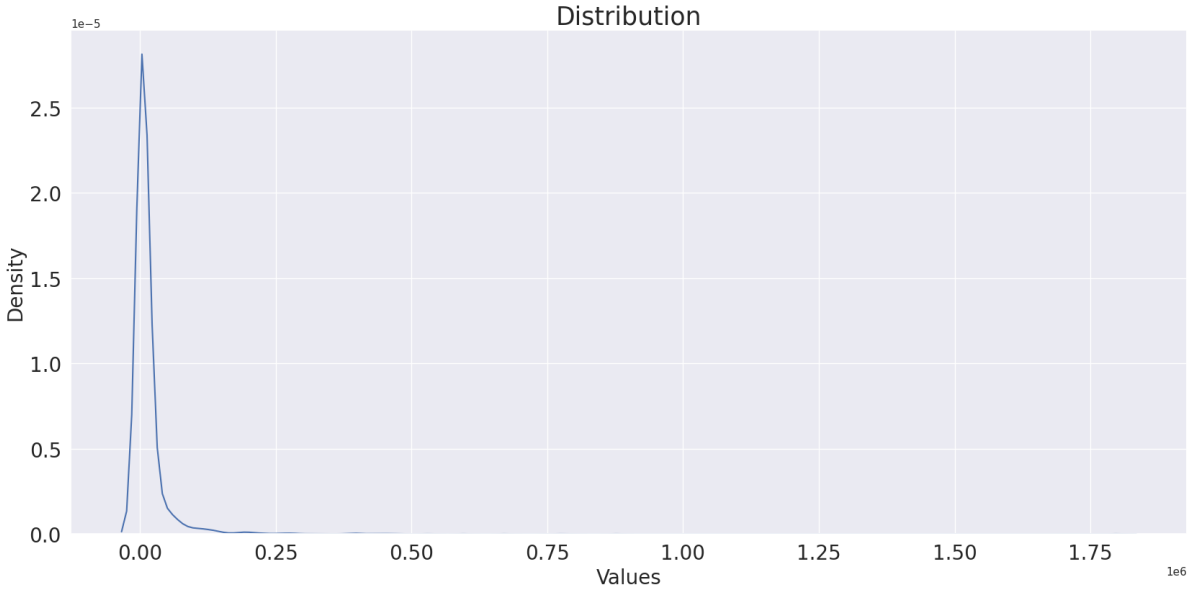
In [ ]:
```
Avg_Ann_Death_data=copy.deepcopy(merged_data.drop(['Incidence Rate','Avg Ann Incide
Incidence_Rate_data=copy.deepcopy(merged_data.drop(['Avg Ann Deaths','Avg Ann Incic
Avg_Ann_Incidence_data=copy.deepcopy(merged_data.drop(['Incidence Rate','Avg Ann De
Mortality_Rate_data=copy.deepcopy(merged_data.drop(['Incidence Rate','Avg Ann Incic
```

Get the outliers and data distribution

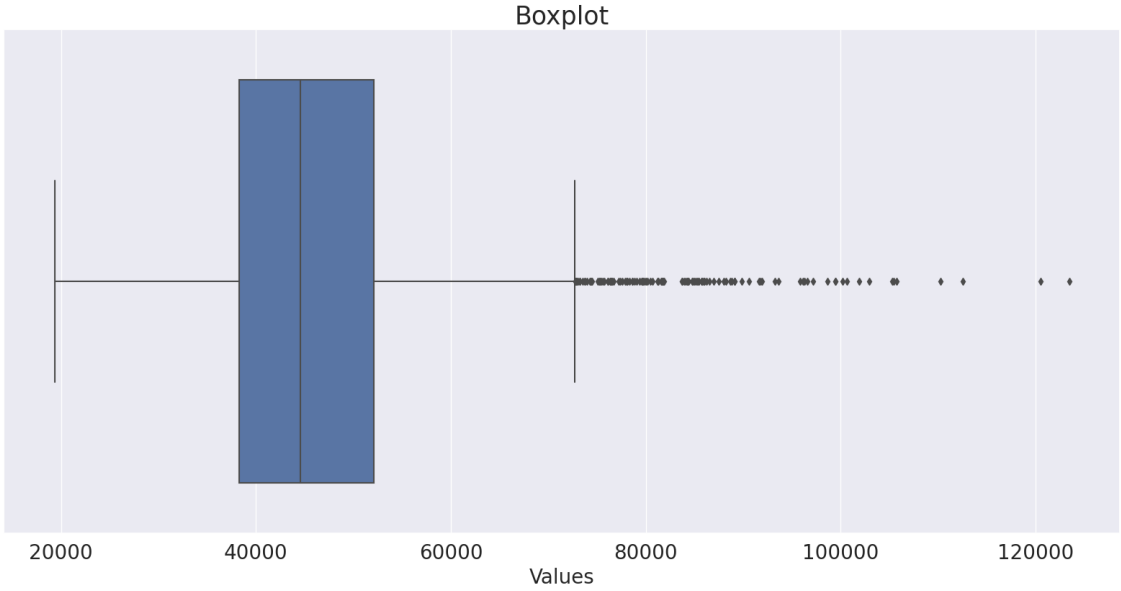
In [ ]:
```
get_box_plots(Avg_Ann_Death_data,save=True)
```

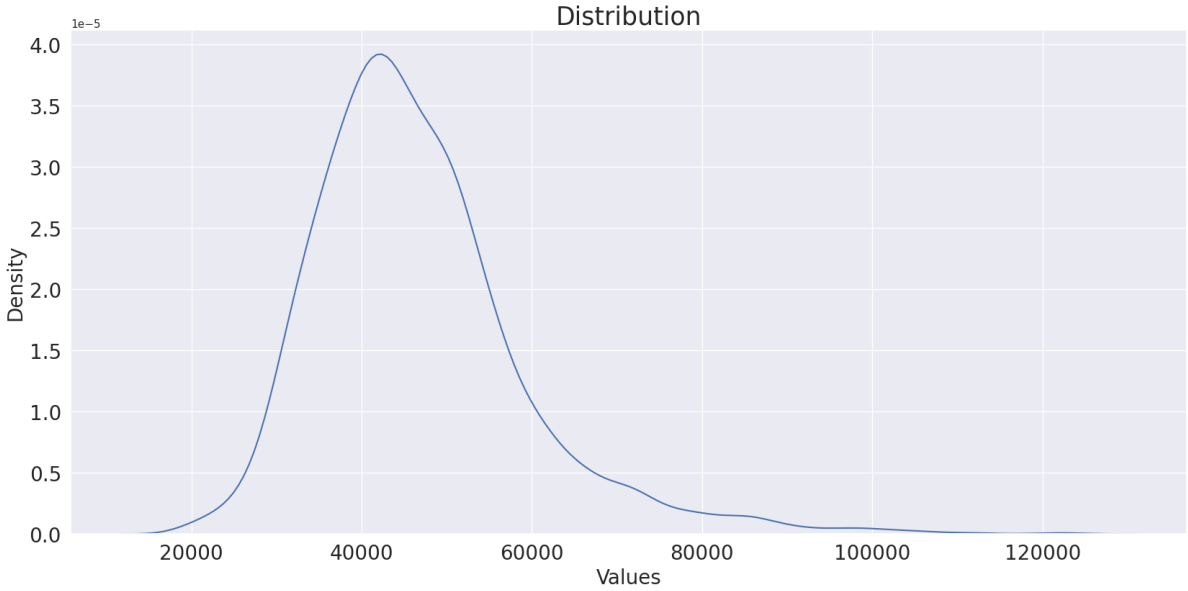
Box and distribution plot for All Poverty



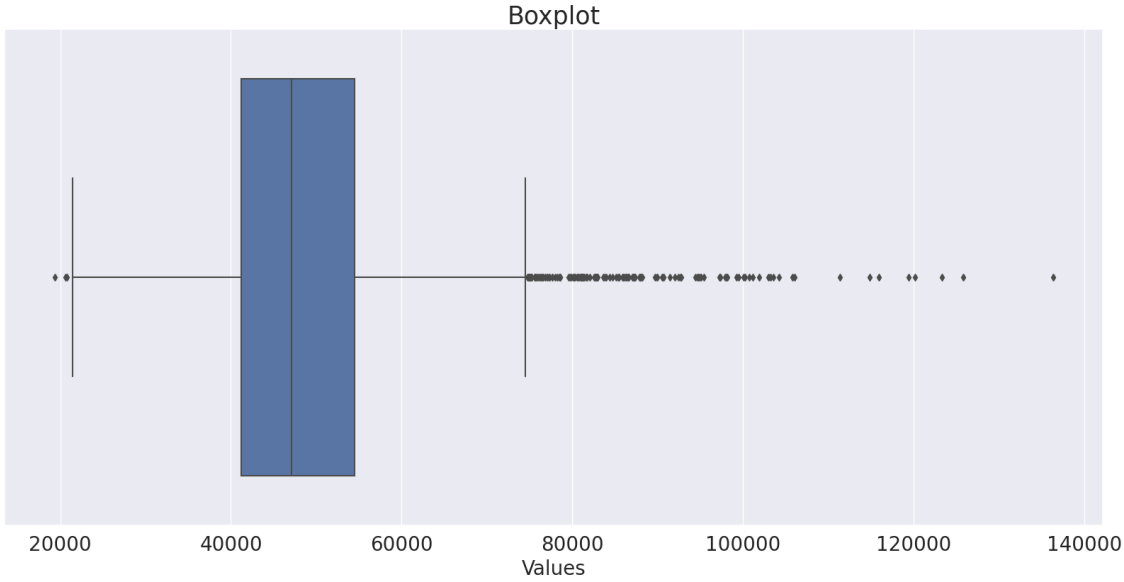


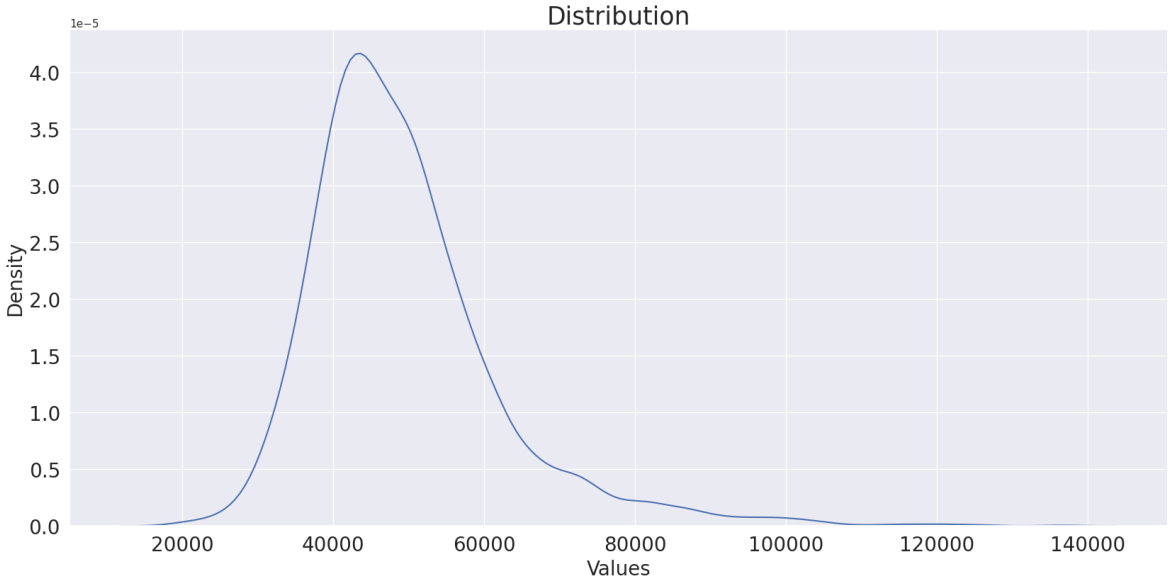
Box and distribution plot for Med Income



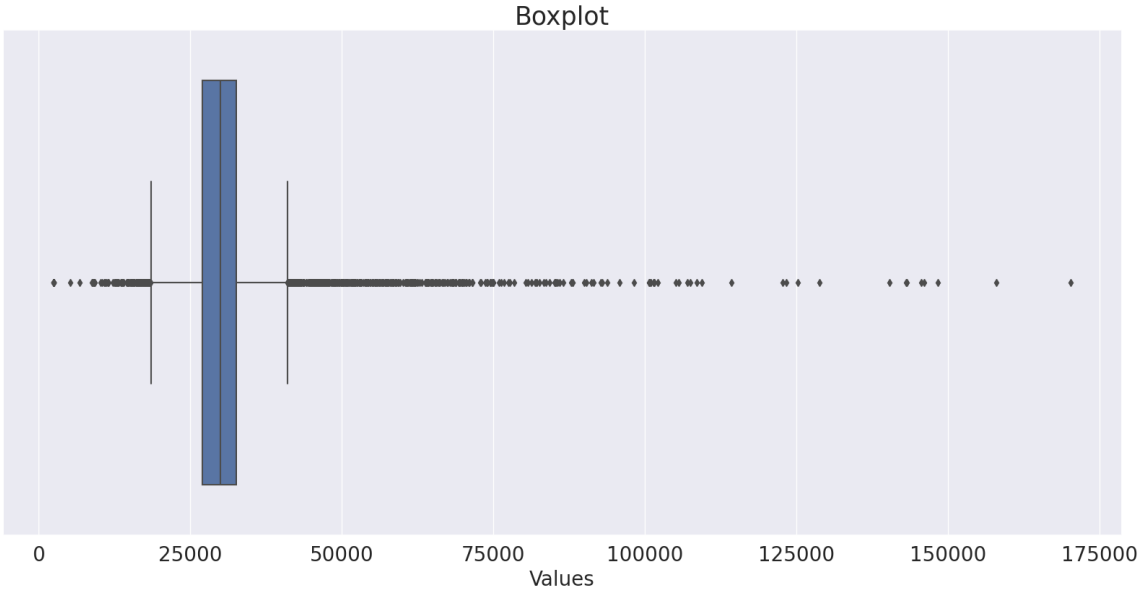


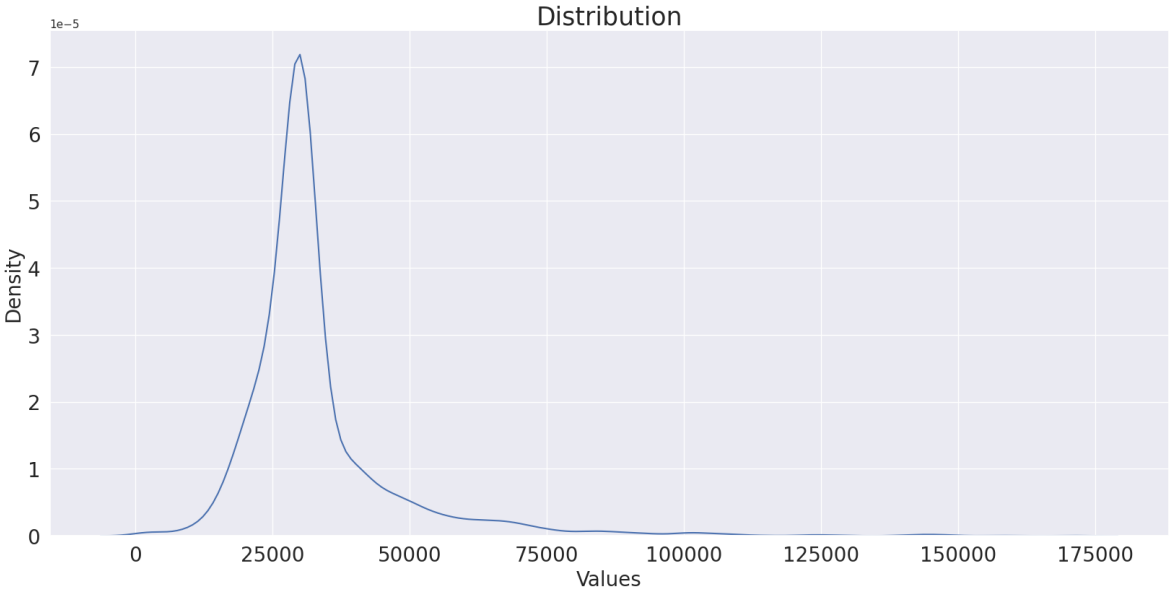
Box and distribution plot for Med Income White



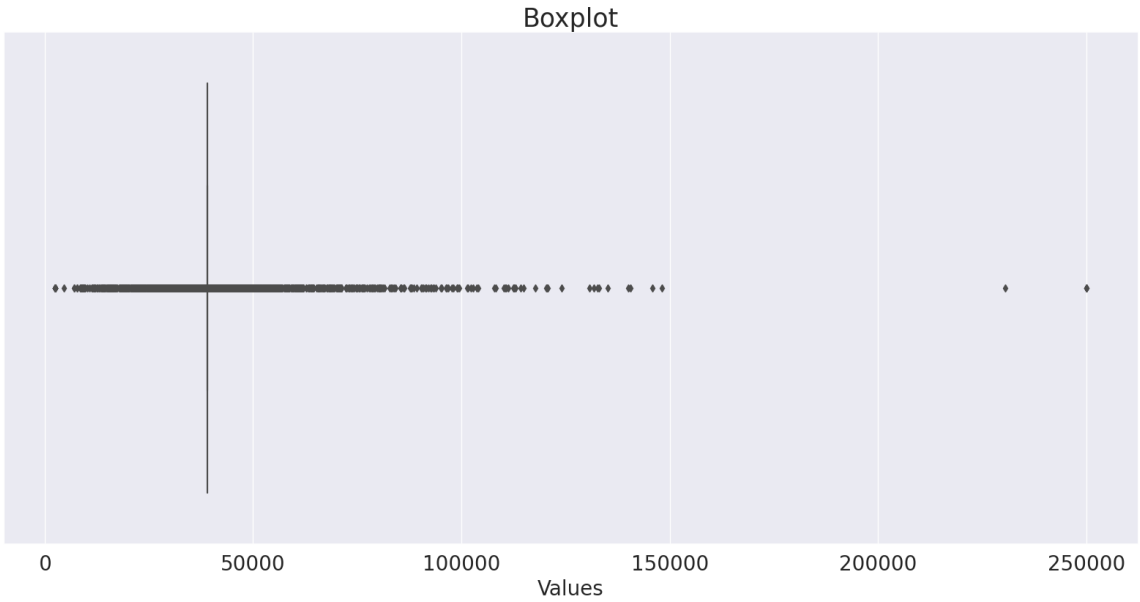


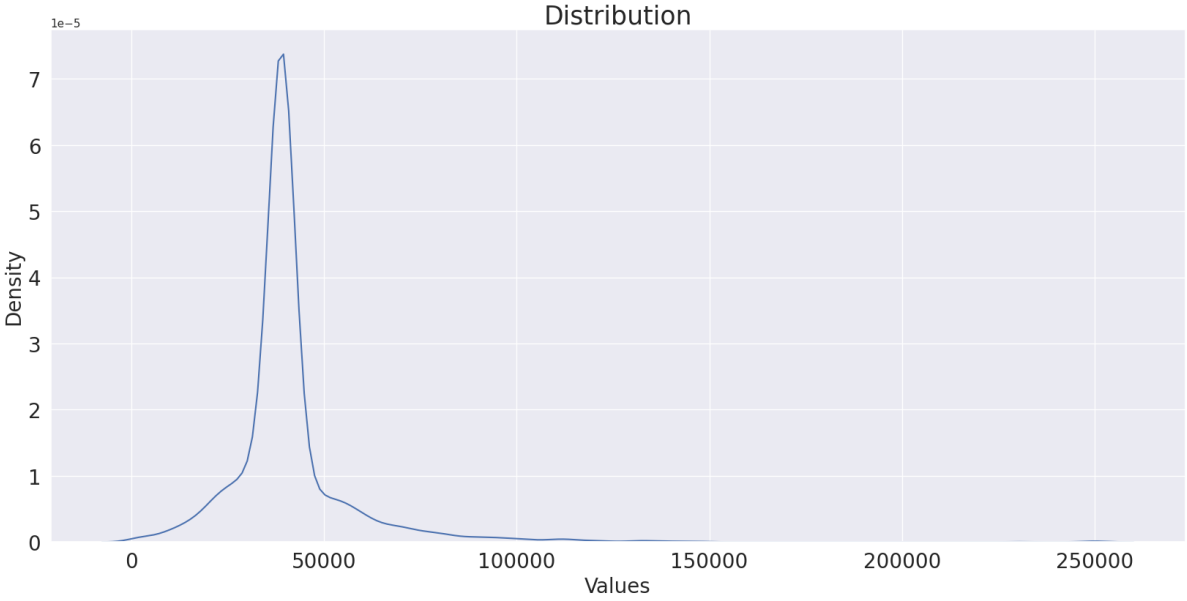
Box and distribution plot for Med Income Black



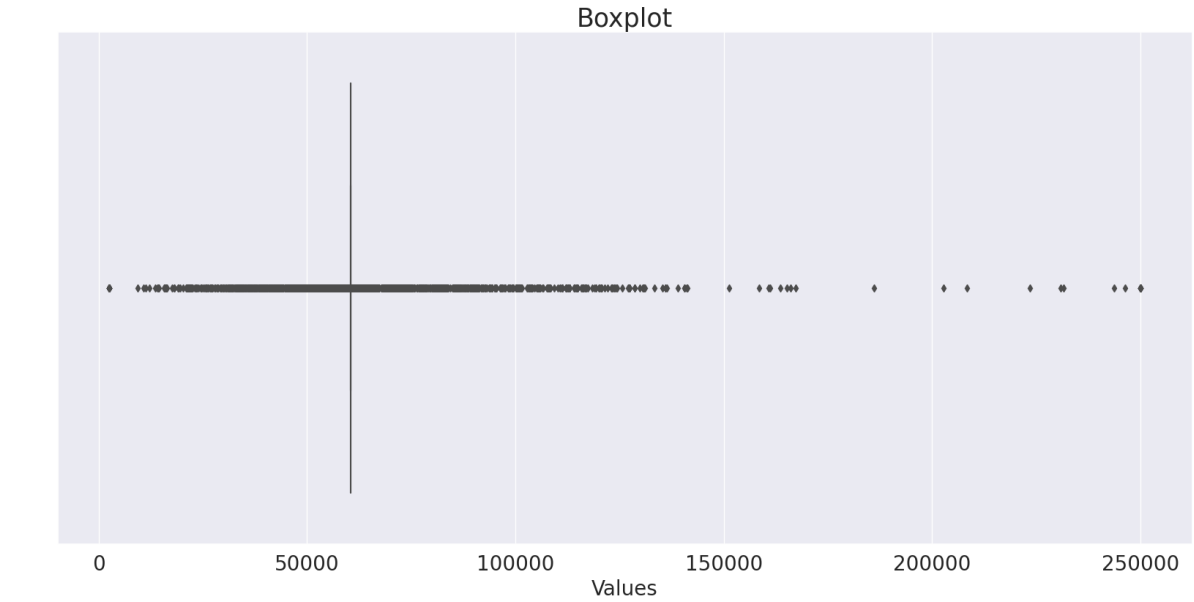


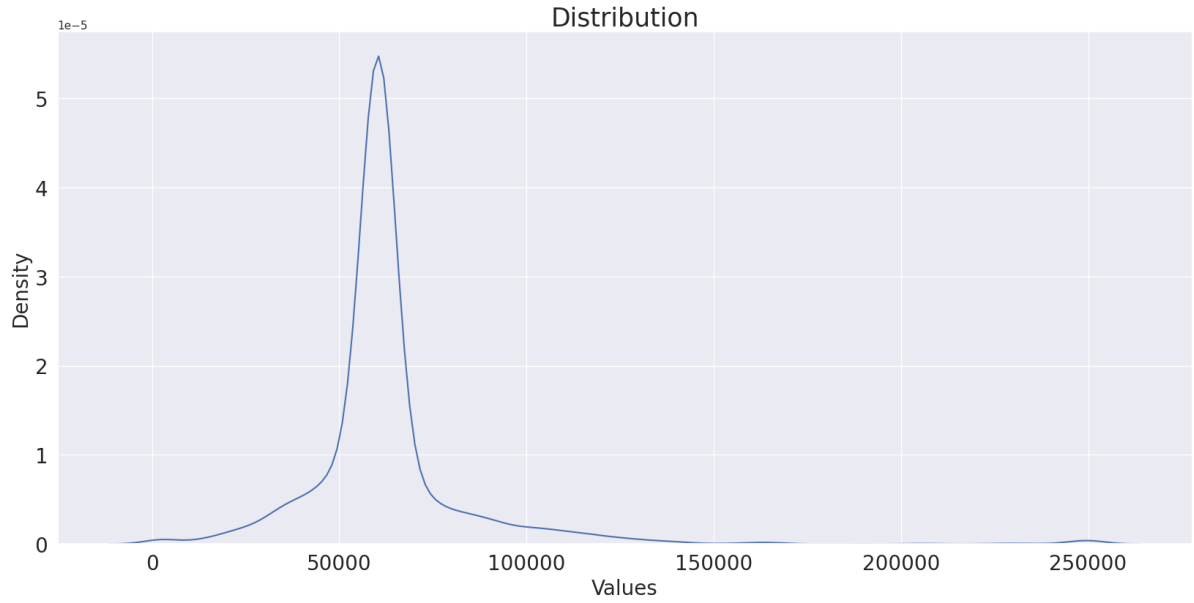
Box and distribution plot for Med Income Nat Am



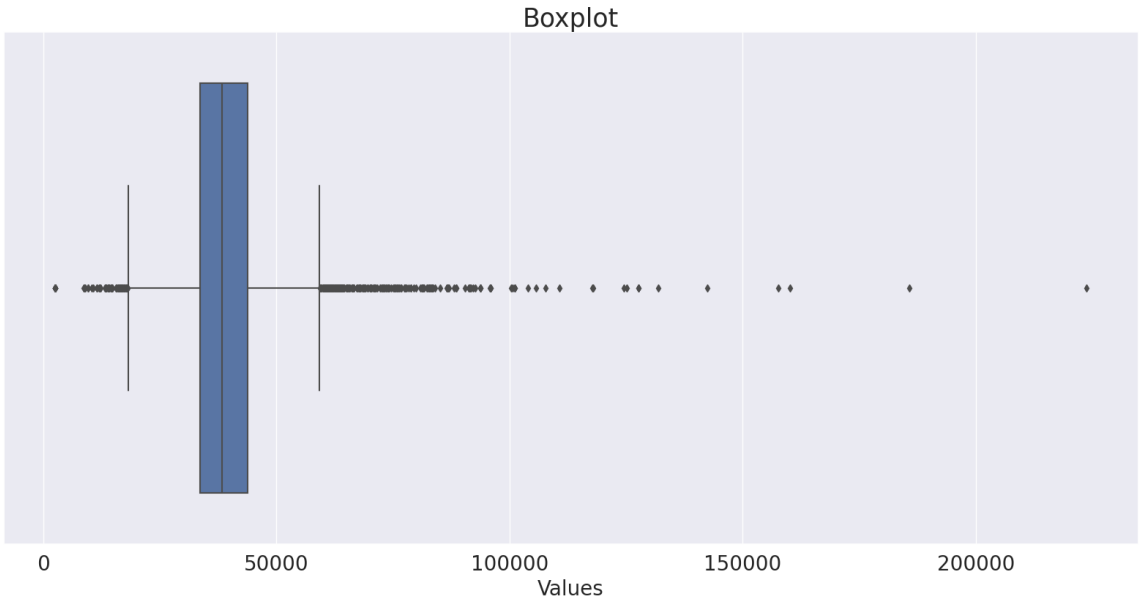


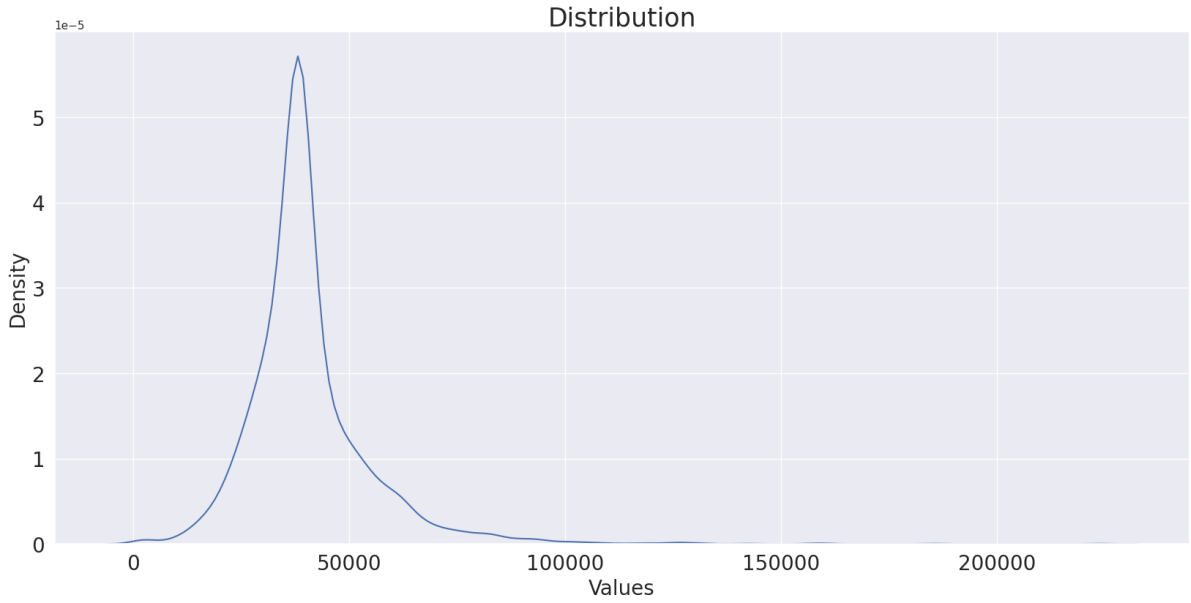
Box and distribution plot for Med Income Asian



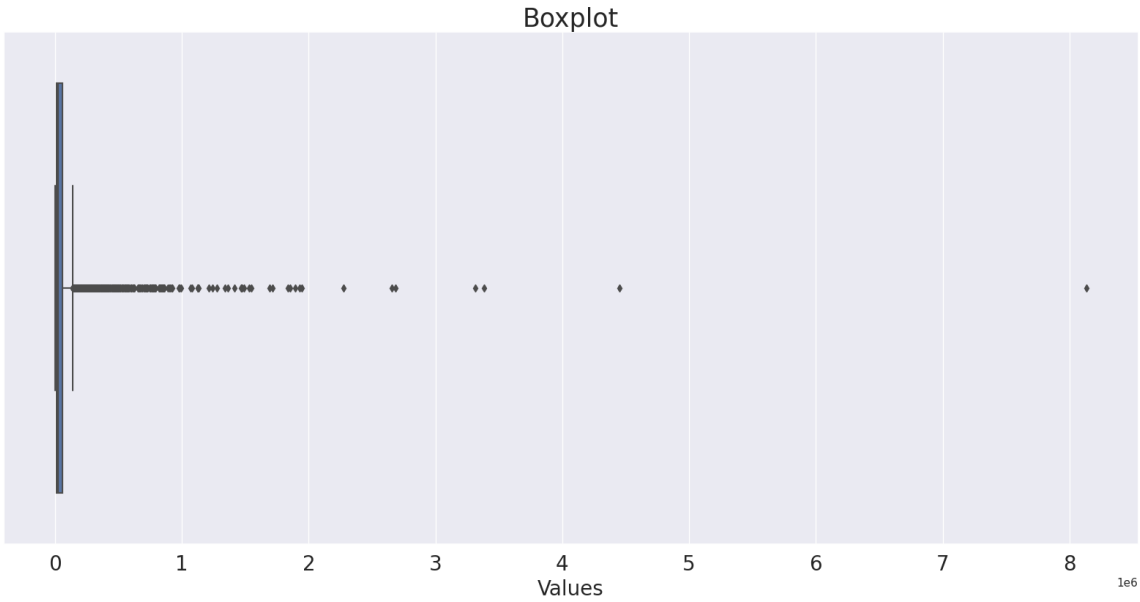


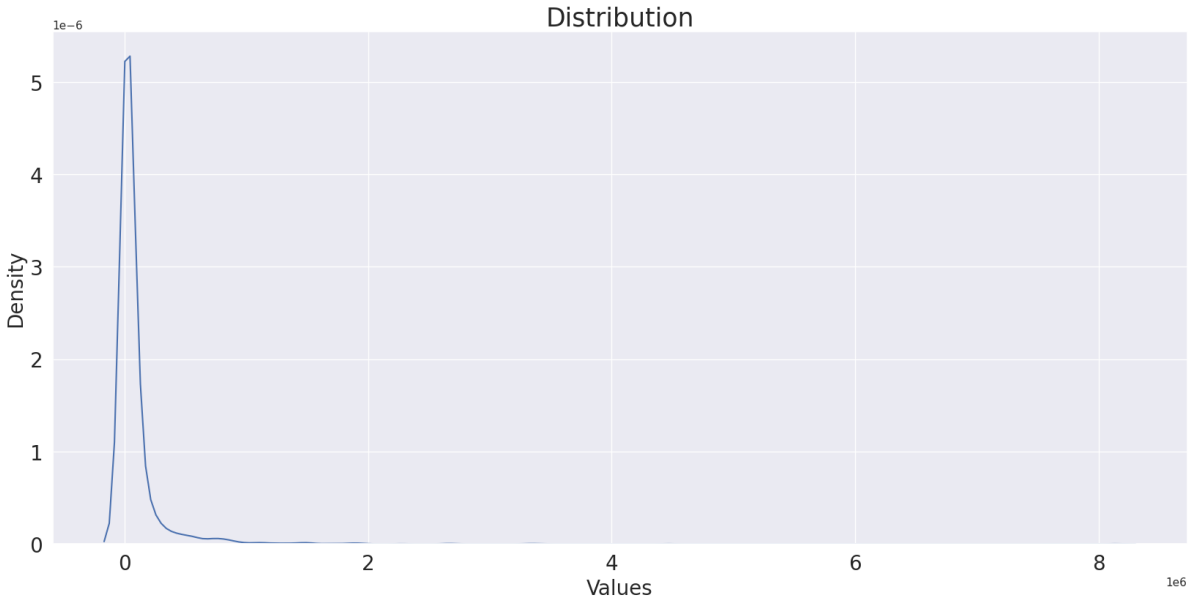
Box and distribution plot for Hispanic



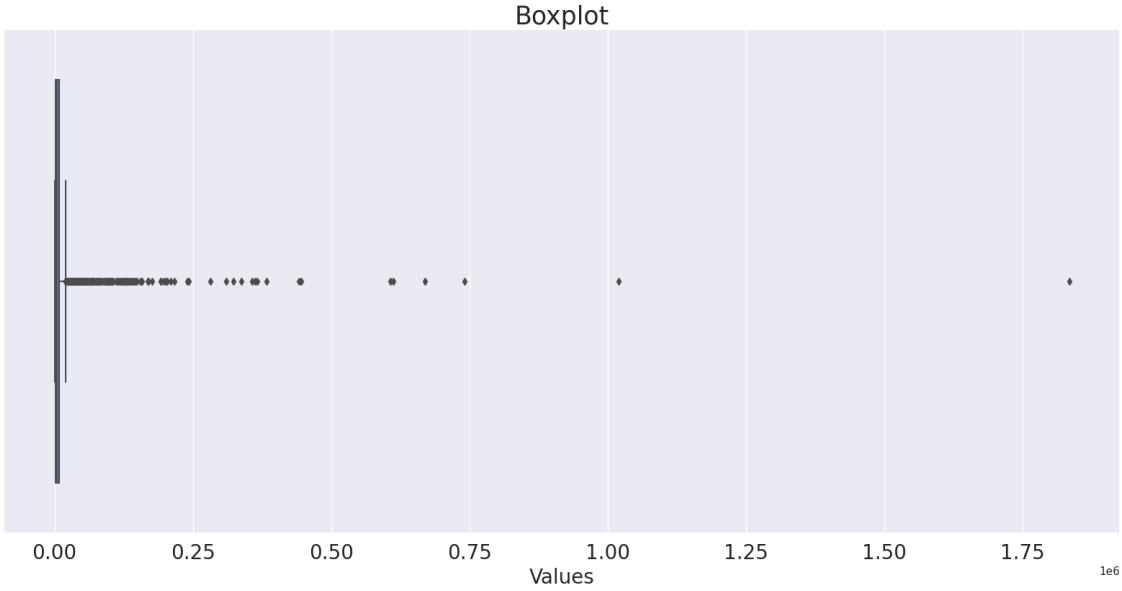


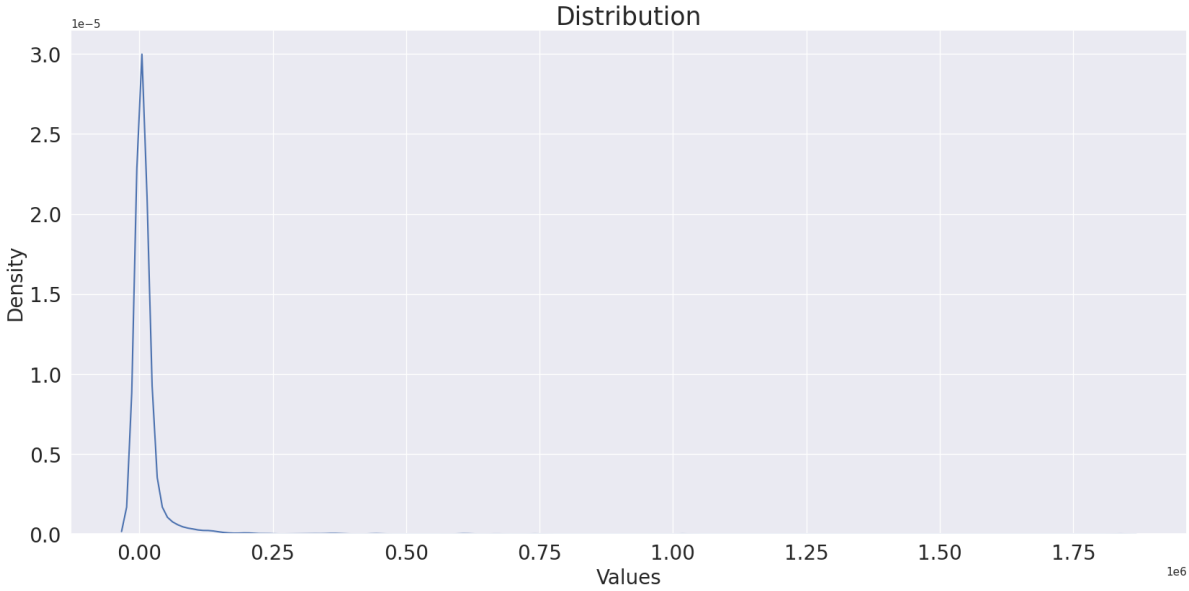
Box and distribution plot for All With

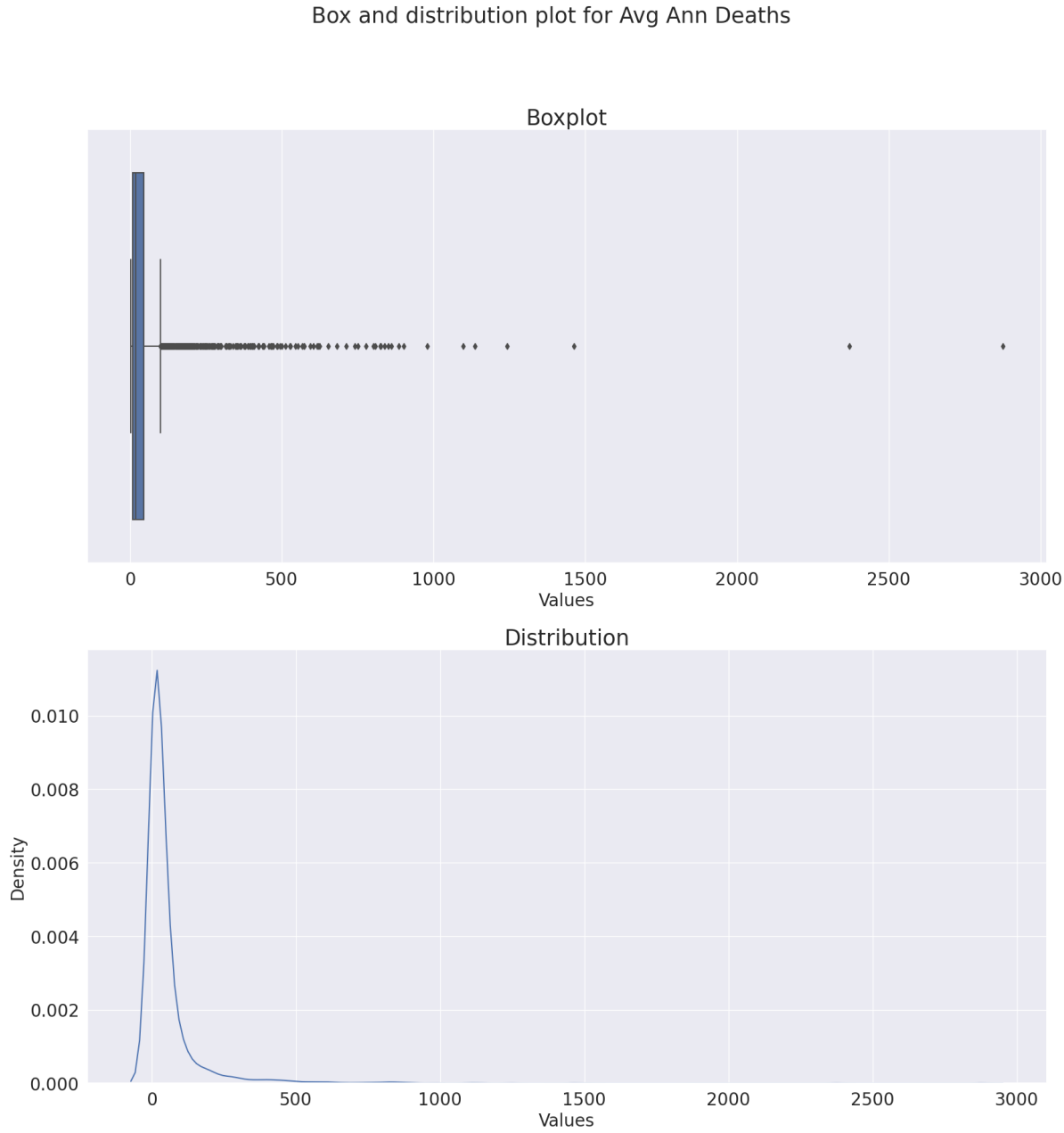




Box and distribution plot for All Without







In [ ]:

```
def Outlier_remove(data,thresh=1.5):
  """
  eliminates the outlier at a given threshold of standard deviation default is 1.5
  data : a dataframe

  """
  z=z=np.abs(stats.zscore(data))#get the z score
  final_data = data[(z < thresh).all(axis=1)]
  return final_data
```

Drop the outliers lying above 1.5σ away from mean. Note we tried with several values of thresholds. Keeping small threshold resulted complete removal of outliers at the cost of loosing out training data. Model trained on less data causes overfitting. With too many outliers on the other hand causes model to make false prediction. So with all this 1.5 turned out to be the best choice

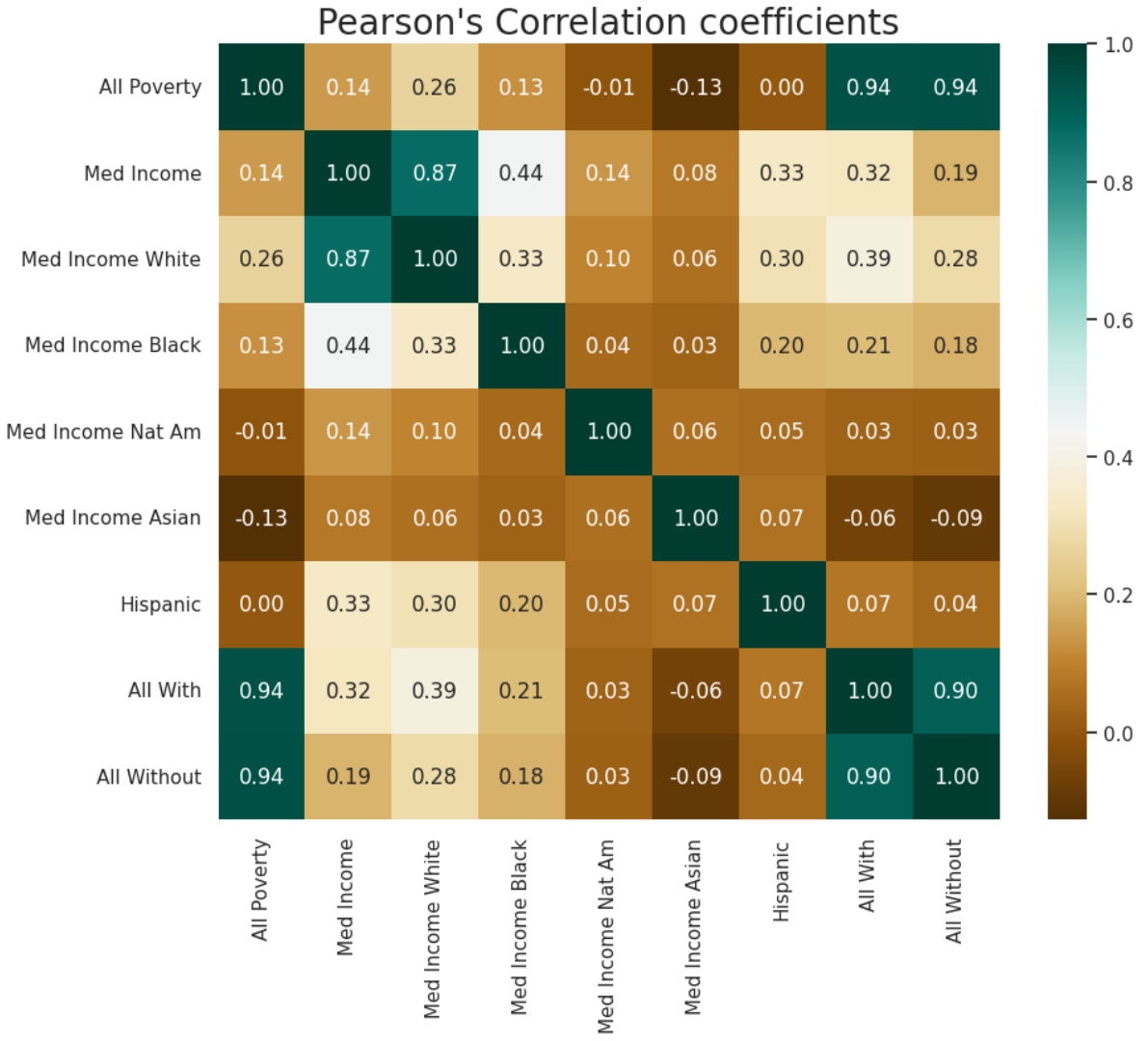
In [ ]:
```
Avg_Ann_Death_wo = Outlier_remove(Avg_Ann_Death_data,1.5)#Remove the outliers
```

Extract target column Avg_Ann_Deaths

In [ ]:
```
Avg_Ann_Death_Target=Avg_Ann_Death_wo.pop('Avg Ann Deaths')
```

Final Correlation before model training

In [ ]:
```
get_corr_plot(Avg_Ann_Death_wo)#correlation heatmap
```



In [ ]:
```
#Drop the all poverty column its highly correlated
Avg_Ann_Death_wo=Avg_Ann_Death_wo.drop('All Poverty',axis=1)
```

Model for Avg_Ann_Deaths target

Initial Model

test_split=0.2

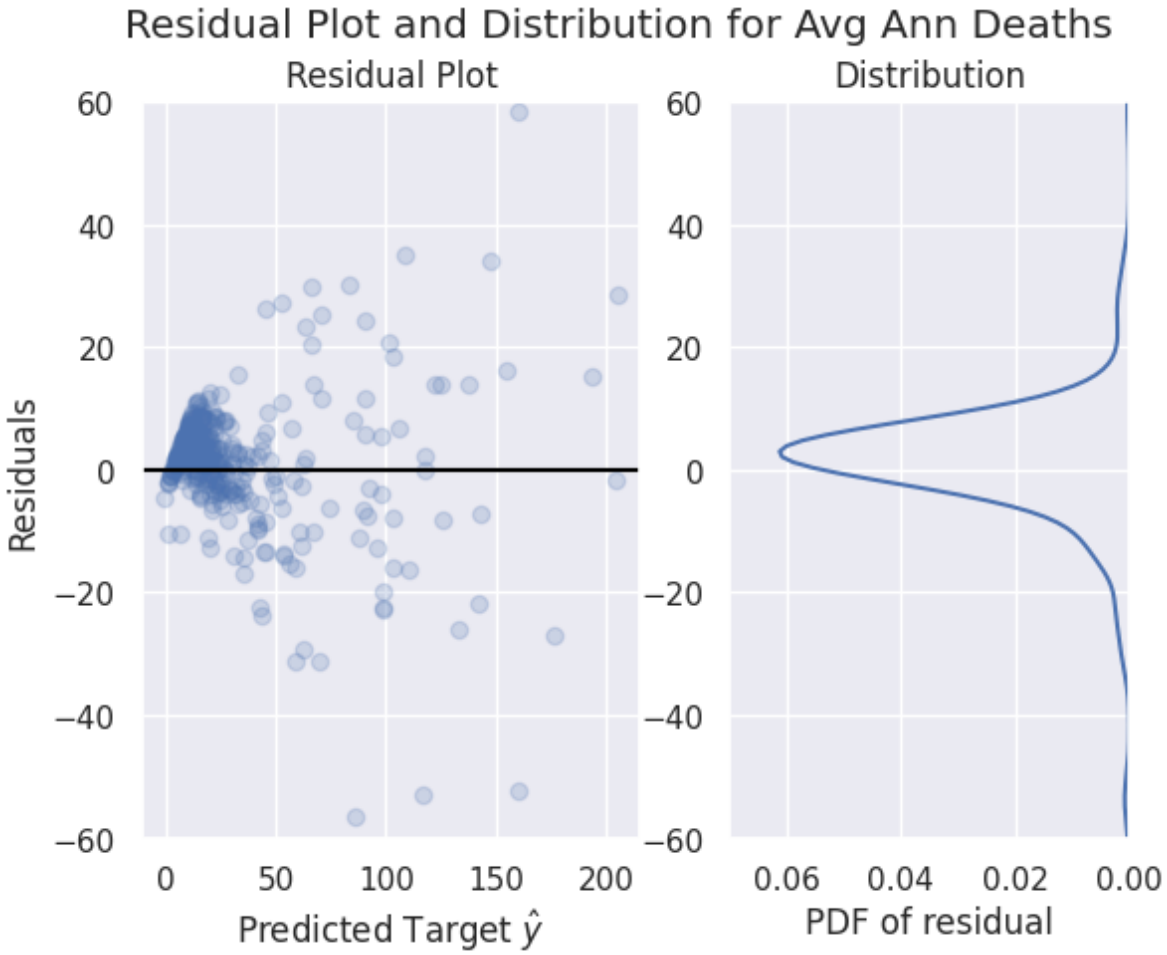
random_state=90

In [ ]:
```
lm0=LinearRegression()
X_train,X_test,y_train,y_test=train_test_split(Avg_Ann_Death_wo,Avg_Ann_Death_Targe
S=StandardScaler()#Standardizing the data to have 0 mean and 1 standard deviation
X_train=S.fit_transform(X_train)#Standardizing the train data to have 0 mean and 1
X_test=S.fit_transform(X_test)#Standardizing the test data to have with mean and st
lm0.fit(X_train,y_train)#training
yhat_test=lm0.predict(X_test)#predict the output
print("R squared score of the model : {}".format(r2_score(y_test,yhat_test)))#gives
```

```
R squared score of the model : 0.9121671415824305
```

Model Evaluation by residual plot

In [ ]:
```
get_residual_plot(y_test,yhat_test,col="Avg Ann Deaths",save=False)
```



The error is nearly normally distributed with mean nearly 0

Visualizing which input feature is contributing largest towards the output

In [ ]:
```
perm = PermutationImportance(lm0, random_state=0).fit(X_test, y_test)
eli5.show_weights(perm, feature_names = Avg_Ann_Death_wo.columns.tolist())
```

Out[ ]:

Weight	Feature
1.7456 ± 0.1607	All With
0.0083 ± 0.0047	Med Income White
0.0030 ± 0.0013	All Without
0.0013 ± 0.0028	Med Income Black
0.0012 ± 0.0020	Med Income Asian
0.0008 ± 0.0006	Med Income
0.0001 ± 0.0006	Hispanic
0.0000 ± 0.0003	Med Income Nat Am

Extracting the most important feature (All_With) to obtain a univariate linear regression for best fit visualisation

In [ ]:
```
X_new_train=np.zeros((X_train.shape[0],1))
X_new_test=np.zeros((X_test.shape[0],1))
X_new_train[:,0]=X_train[:,6]#6th coulumn is All_With
X_new_test[:,0]=X_test[:,6]#6th coulumn is All_With
```

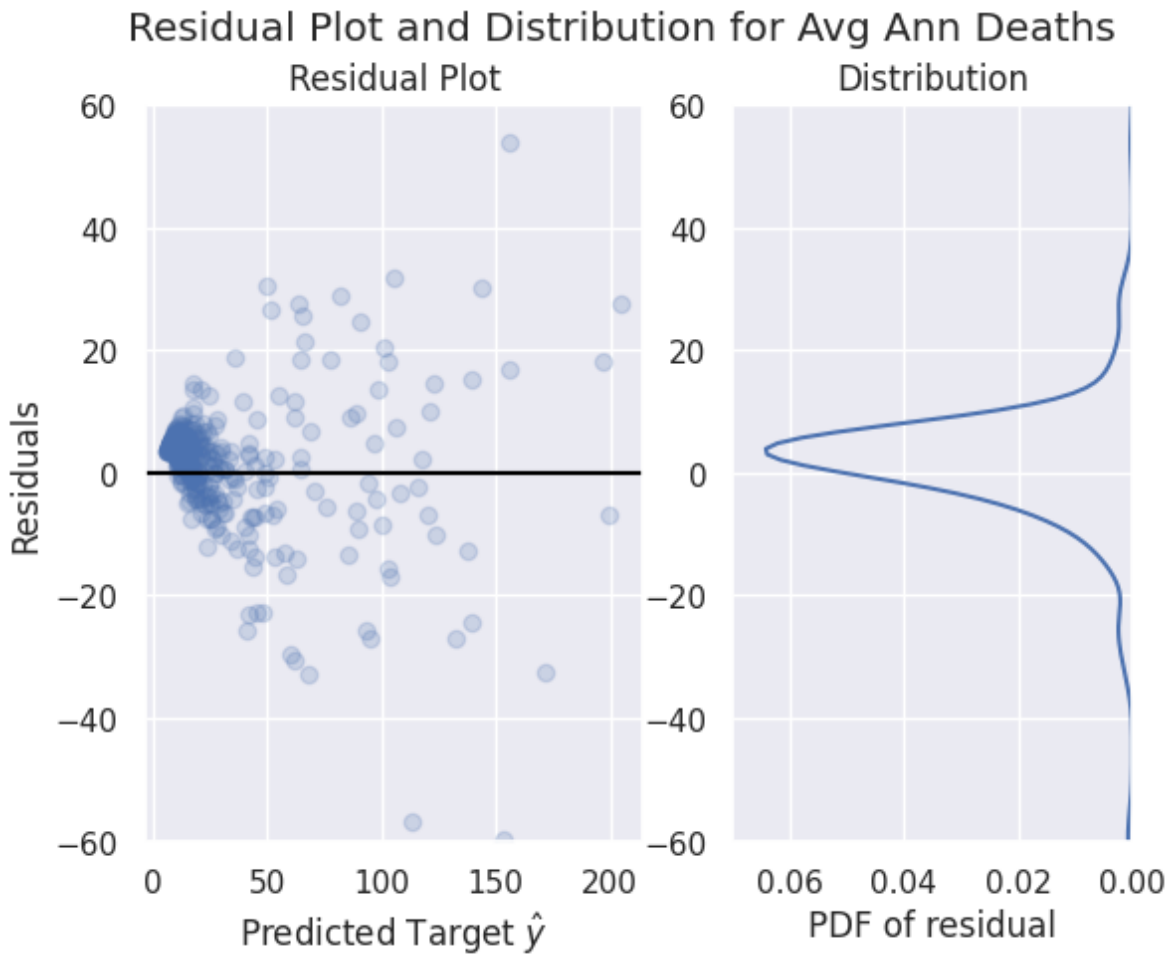
Univariate model

In [ ]:
```
lm0_new=LinearRegression()
lm0_new.fit(X_new_train,y_train)
yhat_new_train=lm0_new.predict(X_new_train)
yhat_new_test=lm0_new.predict(X_new_test)
print("R squared score of the model : {}".format(r2_score(y_test,yhat_new_test)))#g
```

```
R squared score of the model : 0.9060336085071531
```

Residual plot for univariate model

In [ ]:
```
get_residual_plot(y_test,yhat_new_test,col="Avg Ann Deaths",save=False)
```

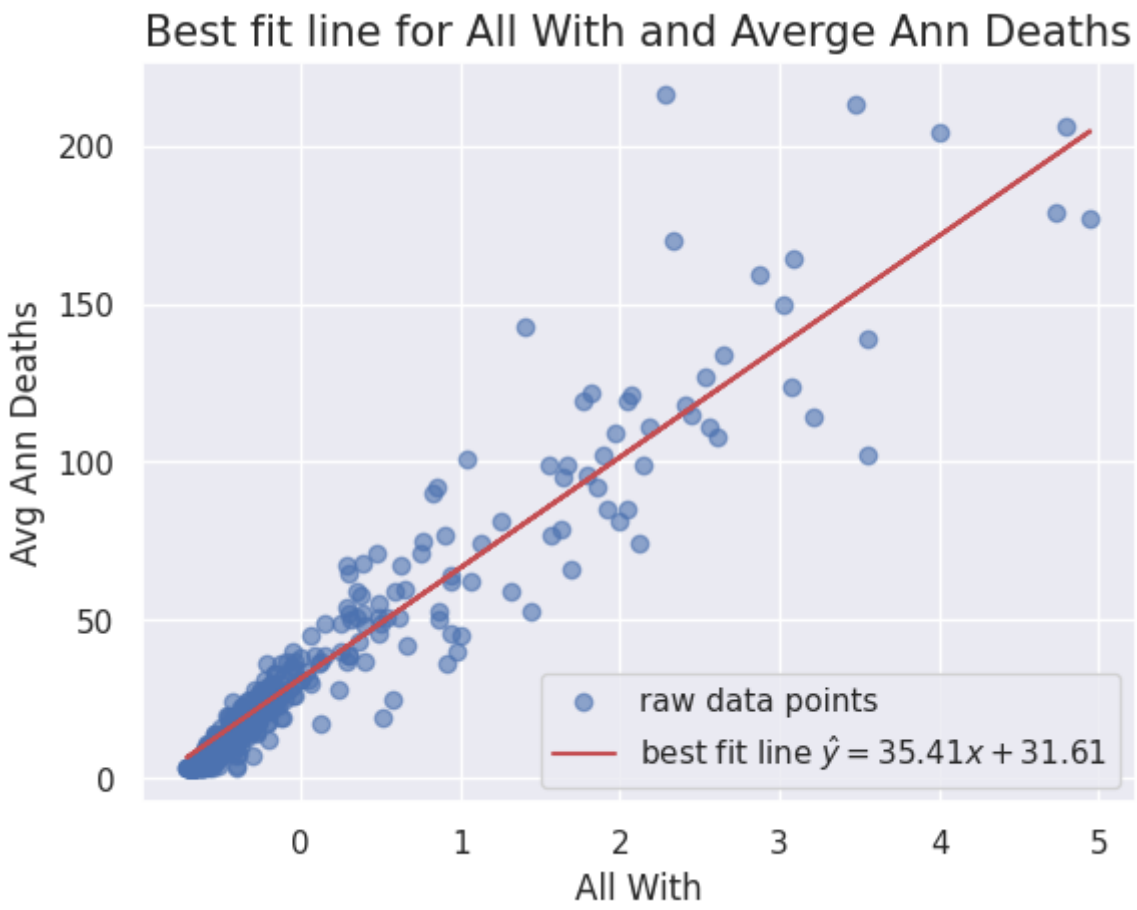


get the model weight (w) and bias (b)

In [ ]:
```
print(f"w: {lm0_new.coef_[0]}")
print(f"b: {lm0_new.intercept_}")
```

```
w: 34.9978476017552
b: 31.5
```

In [ ]:
```
y_line=lm0_new.coef_[0]*np.sort(X_new_test)+lm0_new.intercept_
plt.scatter(X_new_test,y_test,alpha=0.6,label='raw data points')
plt.plot(np.sort(X_new_test),y_line,color='r',label='best fit line $\hat{y}=35.41x+
plt.xlabel("All With")
plt.ylabel("Avg Ann Deaths")
plt.legend()
plt.title("Best fit line for All With and Averge Ann Deaths",fontsize=15)
plt.show()
```



Model for Incidence Rate

In [ ]:
```
Incidence_Rate_data_wo=Outlier_remove(Incidence_Rate_data)#Remove the outliers
Incidence_Rate_data_wo=Incidence_Rate_data_wo.drop('All Poverty',axis=1)
```

In [ ]:
```
Incid_Target=Incidence_Rate_data_wo.pop('Incidence Rate')#get the target
```

In [ ]:
```
lm1=LinearRegression()
X_train,X_test,y_train,y_test=train_test_split(Incidence_Rate_data_wo,Incid_Target,
S=StandardScaler()#Standardizing the data to have 0 mean and 1 standard deviation
X_train=S.fit_transform(X_train)#Standardizing the train data to have 0 mean and 1
X_test=S.fit_transform(X_test)#Standardizing the test data to have with mean and st
lm1.fit(X_train,y_train)#training
yhat_test=lm1.predict(X_test)#predict the output
print("R squared score of the model : {}".format(r2_score(y_test,yhat_test)))#gives
```

```
R squared score of the model : 0.08762430434423296
```

As expected Extremely low R^2 score. No need of further visualisation

Model for Avg_Ann_Incidence

In [ ]:
```
Avg_Ann_Incidence_data_wo=Outlier_remove(Avg_Ann_Incidence_data)
Avg_Ann_Incidence_data_wo=Avg_Ann_Incidence_data_wo.drop("All Poverty",axis=1)
```

In [ ]:
```
Avg_Ann_Incid_Target=Avg_Ann_Incidence_data_wo.pop('Avg Ann Incidence')
```

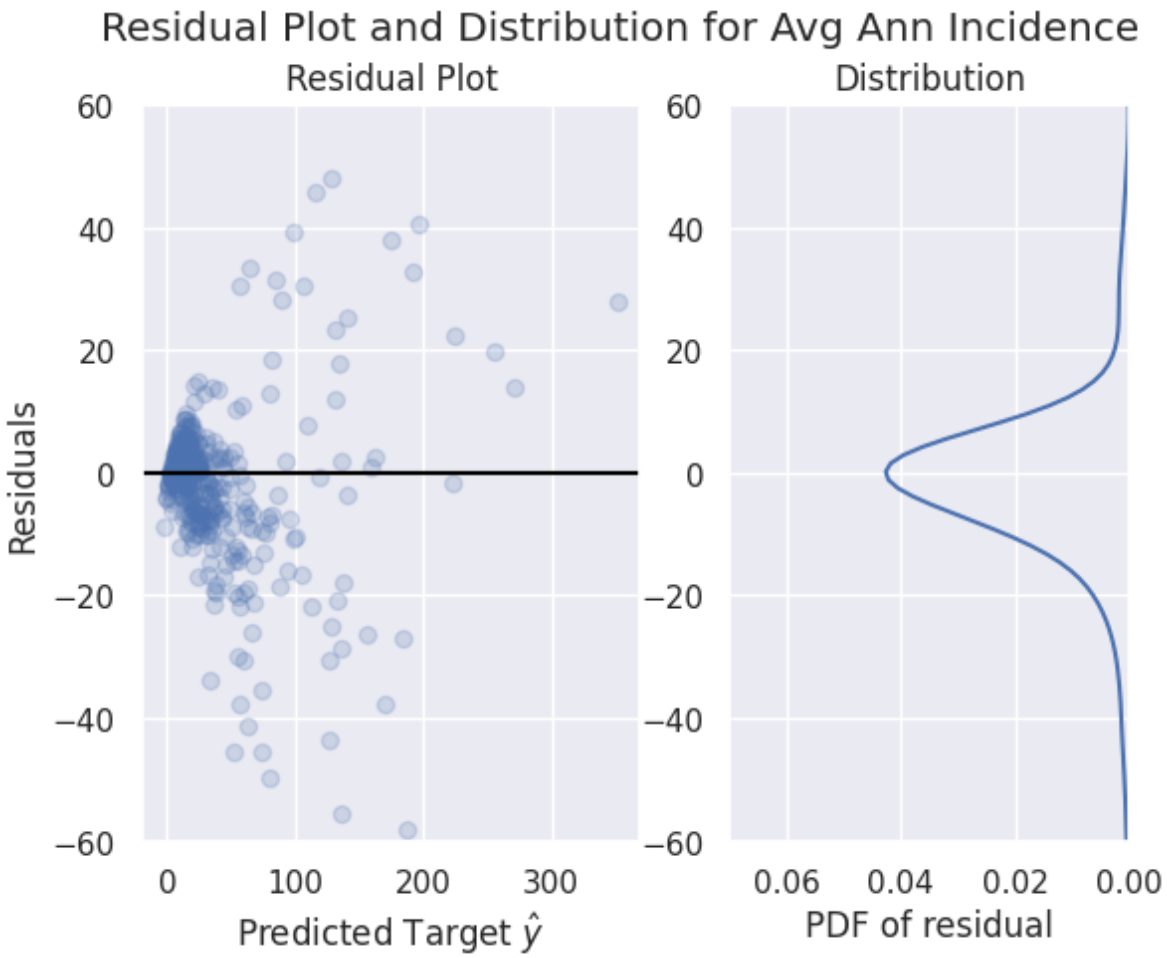
Initial Model

In [ ]:
```
lm2=LinearRegression()
X_train,X_test,y_train,y_test=train_test_split(Avg_Ann_Incidence_data_wo,Avg_Ann_Ir
S=StandardScaler()#Standardizing the data to have 0 mean and 1 standard deviation
X_train=S.fit_transform(X_train)#Standardizing the train data to have 0 mean and 1
X_test=S.fit_transform(X_test)#Standardizing the test data to have with mean and st
lm2.fit(X_train,y_train)#training
yhat_test=lm2.predict(X_test)#predict the output
print("R squared score of the model : {}".format(r2_score(y_test,yhat_test)))#gives
```

```
R squared score of the model : 0.8918948363362549
```

Residual plot of original model

In [ ]:
```
get_residual_plot(y_test,yhat_test,col="Avg Ann Incidence",save=False)
```



Visualizing which input feature is contributing largest towards the output

In [ ]:
```
perm = PermutationImportance(lm2, random_state=0).fit(X_test, y_test)
eli5.show_weights(perm, feature_names = Avg_Ann_Incidence_data_wo.columns.tolist())
```

Out[ ]:

Weight	Feature
1.7384 ± 0.1217	All With
0.0100 ± 0.0048	Med Income White
0.0016 ± 0.0010	All Without
0.0014 ± 0.0037	Med Income Black
0.0006 ± 0.0007	Med Income Asian
0.0006 ± 0.0007	Hispanic
0.0001 ± 0.0005	Med Income Nat Am
0.0001 ± 0.0010	Med Income

Here also All_With is most important feature

In [ ]:
```
X_new_train=np.zeros((X_train.shape[0],1))
X_new_test=np.zeros((X_test.shape[0],1))
X_new_train[:,0]=X_train[:,6]#6th coulumn is All_With
X_new_test[:,0]=X_test[:,6]#6th coulumn is All_With
```

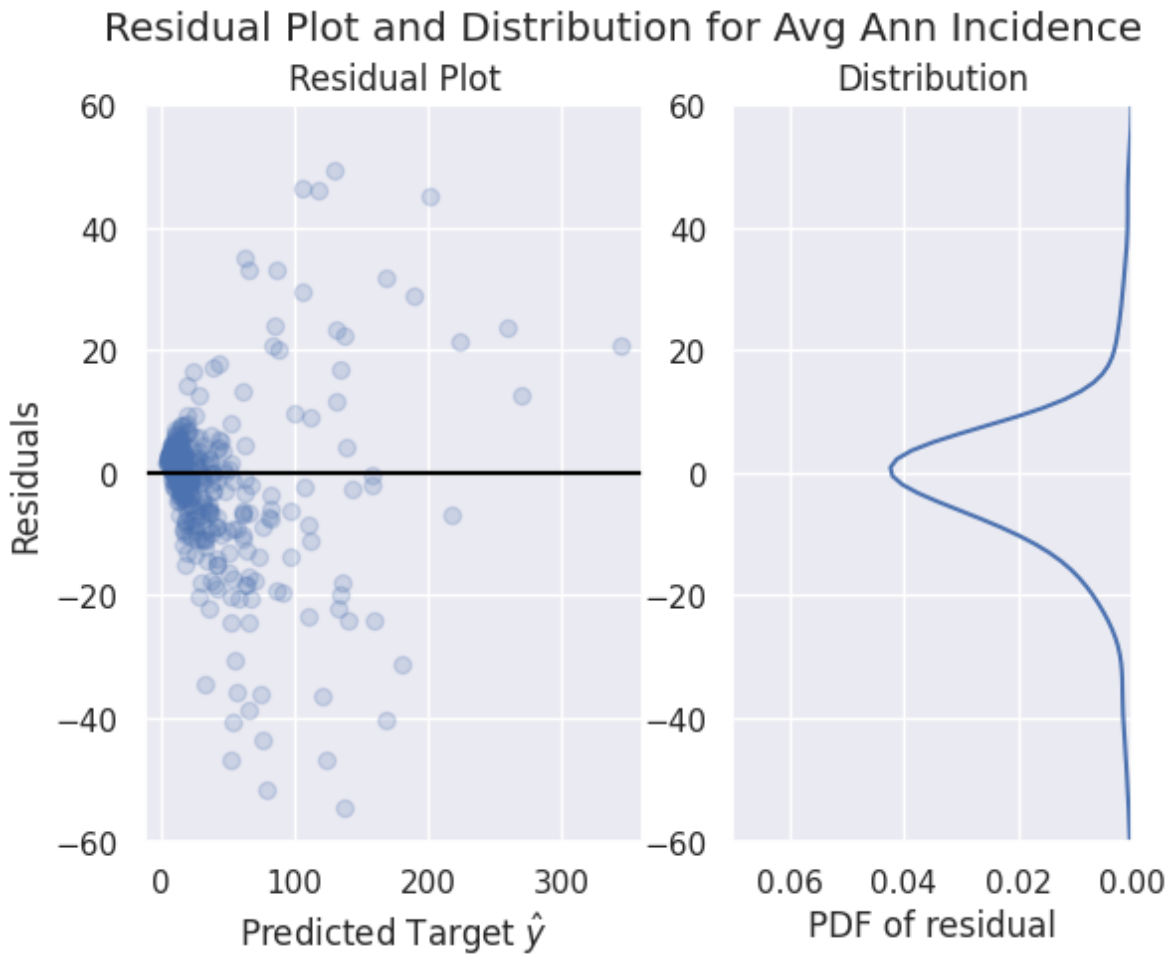
Univariate model

In [ ]:
```
lm2_new=LinearRegression()
lm2_new.fit(X_new_train,y_train)#training
yhat_new_test=lm2_new.predict(X_new_test)#predict the output
print("R squared score of the model : {}".format(r2_score(y_test,yhat_new_test)))#g
```

```
R squared score of the model : 0.8841340294843053
```

Residual plot for univariate model

In [ ]:
```
get_residual_plot(y_test,yhat_new_test,col="Avg Ann Incidence",save=False)
```

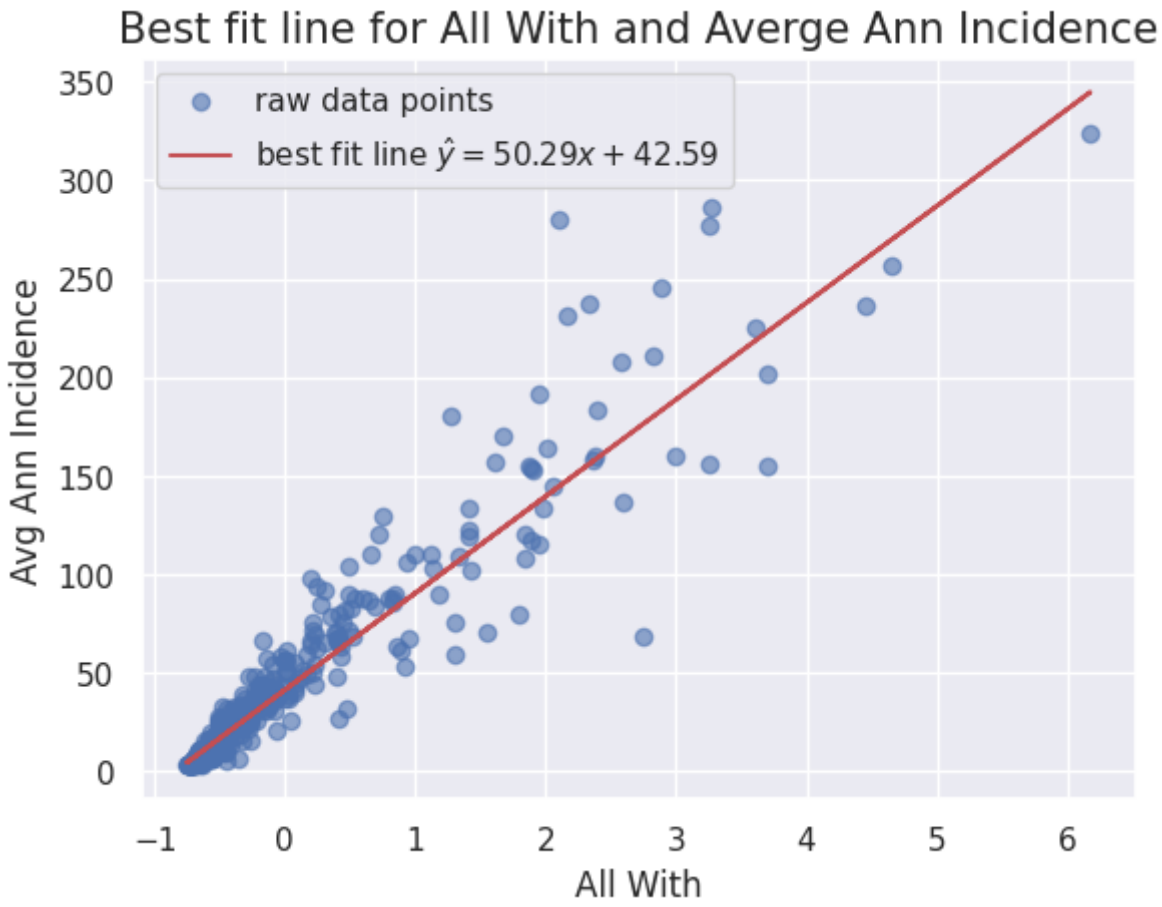


get the model weight (w) and bias (b)

In [ ]:
```
print(f"w: {lm2_new.coef_[0]}")
print(f"b: {lm2_new.intercept_}")
```

```
w: 49.08690848906458
b: 41.407407407407405
```

In [ ]:
```
y_line=lm2_new.coef_[0]*np.sort(X_new_test)+lm2_new.intercept_
plt.scatter(X_new_test,y_test,alpha=0.6,label='raw data points')
plt.plot(np.sort(X_new_test),y_line,color='r',label='best fit line $\hat{y}=50.29x+
plt.xlabel("All With")
plt.ylabel("Avg Ann Incidence")
plt.legend()
plt.title("Best fit line for All With and Averge Ann Incidence",fontsize=15)
plt.show()
```



Model for Mortality Rate

In [ ]:
```
Mortality_Rate_data_wo=Outlier_remove(Mortality_Rate_data)
Mortality_Rate_data_wo=Mortality_Rate_data_wo.drop('All Poverty',axis=1)
Mortality_Rate_data_Target=Mortality_Rate_data_wo.pop('Mortality Rate')
lm3=LinearRegression()
X_train,X_test,y_train,y_test=train_test_split(Mortality_Rate_data_wo,Mortality_Rat
S=StandardScaler()#Standardizing the data to have 0 mean and 1 standard deviation
X_train=S.fit_transform(X_train)#Standardizing the train data to have 0 mean and 1
X_test=S.fit_transform(X_test)#Standardizing the test data to have with mean and st
lm3.fit(X_train,y_train)#training
yhat_test=lm3.predict(X_test)#predict the output
print("R squared score of the model : {}".format(r2_score(y_test,yhat_test)))#gives
```

```
R squared score of the model : 0.17005149498554428
```

Again very poor score

End of code

DAL assignment Logistic Regression (revised)

Importing libraries

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.impute import KNNImputer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import LearningCurveDisplay
from sklearn.metrics import classification_report,confusion_matrix,roc_curve, auc
```

Loading the data

In [ ]:
```
Train=pd.read_excel('/content/drive/MyDrive/DAL dataset/Assignment 2/train.xlsx')
Test=pd.read_excel('/content/drive/MyDrive/DAL dataset/Assignment 2/test.xlsx')
```

In [ ]:
```
Train.describe()# Get the statistical information on Data
```

Out[ ]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Preliminary study on the data

In [ ]:
```
Train.head()
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN

In [ ]:
```
Train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Has Age and Cabin column missing entries. Because the data is really small dropping the columns are respective rows is not a good idea. We will try some imputation.

In [ ]:
```
# check missing values in train data
Train.isnull().sum()
```

Out[ ]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [ ]:
```
#Get the column names
Train.columns
```

Out[ ]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [ ]:
```
Train['Pclass'].unique()# can be treated as categorical data, will be handled by Or
```

Out[ ]:
```
array([3, 1, 2])
```

In [ ]:
```
Train['SibSp'].unique()
```

Out[ ]:
```
array([1, 0, 3, 4, 2, 5, 8])
```

In [ ]:
```
Train['Parch'].unique()
```

Out[ ]:
```
array([0, 1, 2, 5, 3, 4, 6])
```

In [ ]:
```
Train['Embarked'].unique()#can be handled by OHE once missing values are replaced
```

Out[ ]:
```
array(['S', 'C', 'Q', nan], dtype=object)
```

In [ ]:
```
Train['Cabin'].unique()
```

Out[ ]:
```
array([nan, 'C85', 'C123', 'E46', 'G6', 'C103', 'D56', 'A6',
       'C23 C25 C27', 'B78', 'D33', 'B30', 'C52', 'B28', 'C83', 'F33',
       'F G73', 'E31', 'A5', 'D10 D12', 'D26', 'C110', 'B58 B60', 'E101',
       'F E69', 'D47', 'B86', 'F2', 'C2', 'E33', 'B19', 'A7', 'C49', 'F4',
       'A32', 'B4', 'B80', 'A31', 'D36', 'D15', 'C93', 'C78', 'D35',
       'C87', 'B77', 'E67', 'B94', 'C125', 'C99', 'C118', 'D7', 'A19',
       'B49', 'D', 'C22 C26', 'C106', 'C65', 'E36', 'C54',
       'B57 B59 B63 B66', 'C7', 'E34', 'C32', 'B18', 'C124', 'C91', 'E40',
       'T', 'C128', 'D37', 'B35', 'E50', 'C82', 'B96 B98', 'E10', 'E44',
       'A34', 'C104', 'C111', 'C92', 'E38', 'D21', 'E12', 'E63', 'A14',
       'B37', 'C30', 'D20', 'B79', 'E25', 'D46', 'B73', 'C95', 'B38',
       'B39', 'B22', 'C86', 'C70', 'A16', 'C101', 'C68', 'A10', 'E68',
       'B41', 'A20', 'D19', 'D50', 'D9', 'A23', 'B50', 'A26', 'D48',
       'E58', 'C126', 'B71', 'B51 B53 B55', 'D49', 'B5', 'B20', 'F G63',
       'C62 C64', 'E24', 'C90', 'C45', 'E8', 'B101', 'D45', 'C46', 'D30',
       'E121', 'D11', 'E77', 'F38', 'B3', 'D6', 'B82 B84', 'D17', 'A36',
       'B102', 'B69', 'E49', 'C47', 'D28', 'E17', 'A24', 'C50', 'B42',
       'C148'], dtype=object)
```

In [ ]:
```
Train['Ticket'].unique()#useless data. it can't give any insight regarding the pred
```

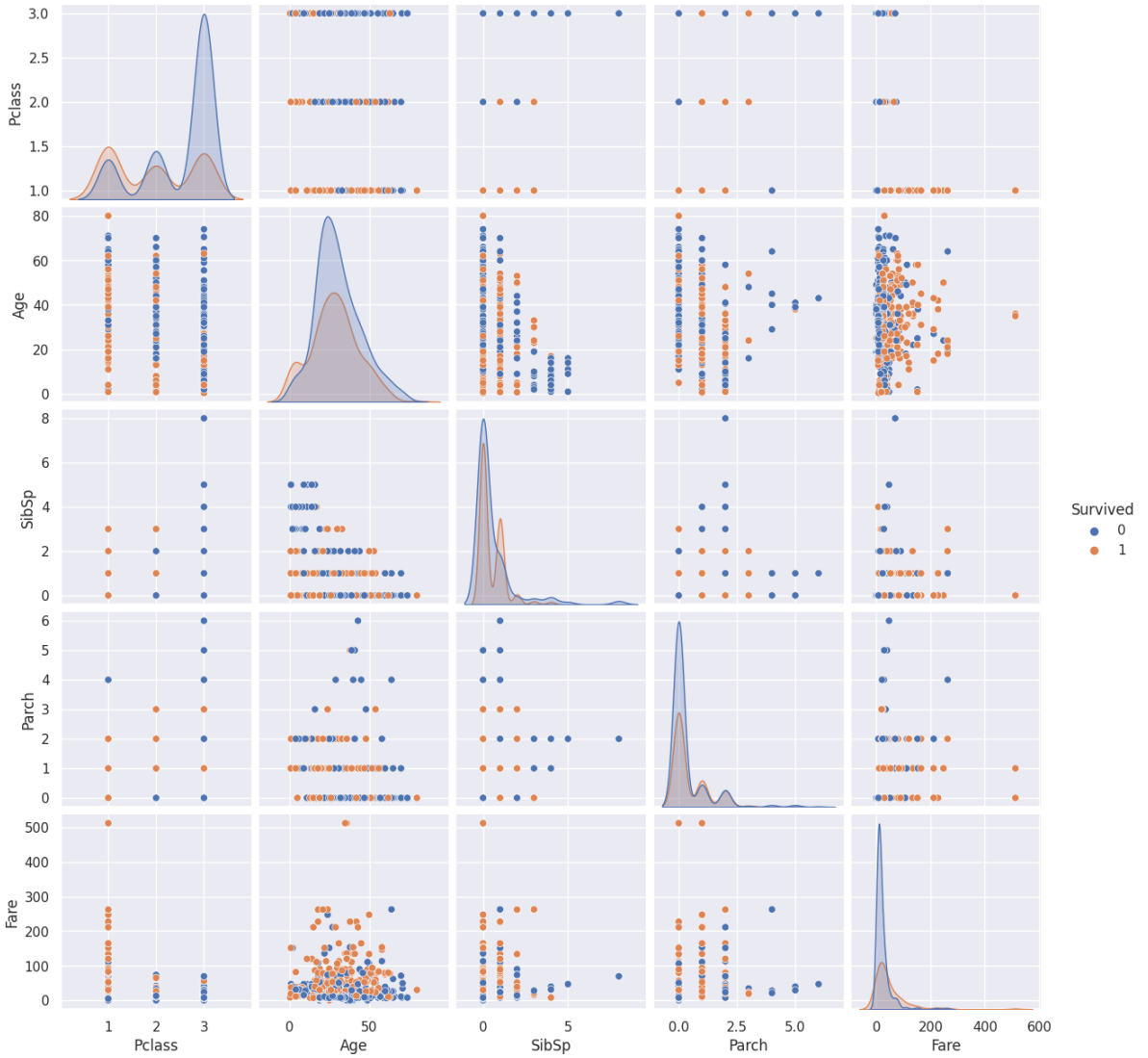
Out[ ]:
```
array(['A/5 21171', 'PC 17599', 'STON/O2. 3101282', 113803, 373450,
       330877, 17463, 349909, 347742, 237736, 'PP 9549', 113783,
       'A/5. 2151', 347082, 350406, 248706, 382652, 244373, 345763, 2649,
       239865, 248698, 330923, 113788, 347077, 2631, 19950, 330959,
       349216, 'PC 17601', 'PC 17569', 335677, 'C.A. 24579', 'PC 17604',
       113789, 2677, 'A./5. 2152', 345764, 2651, 7546, 11668, 349253,
       'SC/Paris 2123', 330958, 'S.C./A.4. 23567', 370371, 14311, 2662,
       349237, 3101295, 'A/4. 39886', 'PC 17572', 2926, 113509, 19947,
       'C.A. 31026', 2697, 'C.A. 34651', 'CA 2144', 2669, 113572, 36973,
       347088, 'PC 17605', 2661, 'C.A. 29395', 'S.P. 3464', 3101281,
       315151, 'C.A. 33111', 'S.O.C. 14879', 2680, 1601, 348123, 349208,
       374746, 248738, 364516, 345767, 345779, 330932, 113059,
       'SO/C 14885', 3101278, 'W./C. 6608', 'SOTON/OQ 392086', 343275,
       343276, 347466, 'W.E.P. 5734', 'C.A. 2315', 364500, 374910,
       'PC 17754', 'PC 17759', 231919, 244367, 349245, 349215, 35281,
       7540, 3101276, 349207, 343120, 312991, 349249, 371110, 110465,
       2665, 324669, 4136, 2627, 'STON/O 2. 3101294', 370369, 'PC 17558',
       'A4. 54510', 27267, 370372, 'C 17369', 2668, 347061, 349241,
       'SOTON/O.Q. 3101307', 'A/5. 3337', 228414, 'C.A. 29178',
       'SC/PARIS 2133', 11752, 7534, 'PC 17593', 2678, 347081,
       'STON/O2. 3101279', 365222, 231945, 'C.A. 33112', 350043, 230080,
       244310, 'S.O.P. 1166', 113776, 'A.5. 11206', 'A/5. 851',
       'Fa 265302', 'PC 17597', 35851, 'SOTON/OQ 392090', 315037,
       'CA. 2343', 371362, 'C.A. 33595', 347068, 315093, 363291, 113505,
       'PC 17318', 111240, 'STON/O 2. 3101280', 17764, 350404, 4133,
       'PC 17595', 250653, 'LINE', 'SC/PARIS 2131', 230136, 315153,
       113767, 370365, 111428, 364849, 349247, 234604, 28424, 350046,
       'PC 17610', 368703, 4579, 370370, 248747, 345770, 3101264, 2628,
       'A/5 3540', 347054, 2699, 367231, 112277, 'SOTON/O.Q. 3101311',
       'F.C.C. 13528', 'A/5 21174', 250646, 367229, 35273,
       'STON/O2. 3101283', 243847, 11813, 'W/C 14208', 'SOTON/OQ 392089',
       220367, 21440, 349234, 19943, 'PP 4348', 'SW/PP 751', 'A/5 21173',
       236171, 347067, 237442, 'C.A. 29566', 'W./C. 6609', 26707,
       'C.A. 31921', 28665, 'SCO/W 1585', 367230, 'W./C. 14263',
       'STON/O 2. 3101275', 2694, 19928, 347071, 250649, 11751, 244252,
       362316, 113514, 'A/5. 3336', 370129, 2650, 'PC 17585', 110152,
       'PC 17755', 230433, 384461, 110413, 112059, 382649, 'C.A. 17248',
       347083, 'PC 17582', 'PC 17760', 113798, 250644, 'PC 17596', 370375,
       13502, 347073, 239853, 'C.A. 2673', 336439, 347464, 345778,
       'A/5. 10482', 113056, 349239, 345774, 349206, 237798, 370373,
       19877, 11967, 'SC/Paris 2163', 349236, 349233, 'PC 17612', 2693,
       113781, 19988, 9234, 367226, 226593, 'A/5 2466', 17421, 'PC 17758',
       'P/PP 3381', 'PC 17485', 11767, 'PC 17608', 250651, 349243,
       'F.C.C. 13529', 347470, 29011, 36928, 16966, 'A/5 21172', 349219,
       234818, 345364, 28551, 111361, 113043, 'PC 17611', 349225, 7598,
       113784, 248740, 244361, 229236, 248733, 31418, 386525,
       'C.A. 37671', 315088, 7267, 113510, 2695, 2647, 345783, 237671,
       330931, 330980, 'SC/PARIS 2167', 2691, 'SOTON/O.Q. 3101310',
       'C 7076', 110813, 2626, 14313, 'PC 17477', 11765, 3101267, 323951,
       'C 7077', 113503, 2648, 347069, 'PC 17757', 2653,
       'STON/O 2. 3101293', 349227, 27849, 367655, 'SC 1748', 113760,
       350034, 3101277, 350052, 350407, 28403, 244278, 240929,
       'STON/O 2. 3101289', 341826, 4137, 315096, 28664, 347064, 29106,
       312992, 349222, 394140, 'STON/O 2. 3101269', 343095, 28220, 250652,
       28228, 345773, 349254, 'A/5. 13032', 315082, 347080, 'A/4. 34244',
       2003, 250655, 364851, 'SOTON/O.Q. 392078', 110564, 376564,
       'SC/AH 3085', 'STON/O 2. 3101274', 13507, 'C.A. 18723', 345769,
       347076, 230434, 65306, 33638, 113794, 2666, 113786, 65303, 113051,
       17453, 'A/5 2817', 349240, 13509, 17464, 'F.C.C. 13531', 371060,
       19952, 364506, 111320, 234360, 'A/S 2816', 'SOTON/O.Q. 3101306',
       113792, 36209, 323592, 315089, 'SC/AH Basle 541', 7553, 31027,
       3460, 350060, 3101298, 239854, 'A/5 3594', 4134, 11771,
       'A.5. 18509', 65304, 'SOTON/OQ 3101317', 113787, 'PC 17609',
       'A/4 45380', 36947, 'C.A. 6212', 350035, 315086, 364846, 330909,
```

```
4135, 26360, 111427, 'C 4001', 382651, 'SOTON/OQ 3101316',
'PC 17473', 'PC 17603', 349209, 36967, 'C.A. 34260', 226875,
349242, 12749, 349252, 2624, 2700, 367232, 'W./C. 14258',
'PC 17483', 3101296, 29104, 2641, 2690, 315084, 113050, 'PC 17761',
364498, 13568, 'WE/P 5735', 2908, 693, 'SC/PARIS 2146', 244358,
330979, 2620, 347085, 113807, 11755, 345572, 372622, 349251,
218629, 'SOTON/OQ 392082', 'SOTON/O.Q. 392087', 'A/4 48871',
349205, 2686, 350417, 'S.W./PP 752', 11769, 'PC 17474', 14312,
'A/4. 20589', 358585, 243880, 2689, 'STON/O 2. 3101286', 237789,
13049, 3411, 237565, 13567, 14973, 'A./5. 3235',
'STON/O 2. 3101273', 'A/5 3902', 364848, 'SC/AH 29037', 248727,
2664, 349214, 113796, 364511, 111426, 349910, 349246, 113804,
'SOTON/O.Q. 3101305', 370377, 364512, 220845, 31028, 2659, 11753,
350029, 54636, 36963, 219533, 349224, 334912, 27042, 347743, 13214,
112052, 237668, 'STON/O 2. 3101292', 350050, 349231, 13213,
'S.O./P.P. 751', 'CA. 2314', 349221, 8475, 330919, 365226, 349223,
29751, 2623, 5727, 349210, 'STON/O 2. 3101285', 234686, 312993,
'A/5 3536', 19996, 29750, 'F.C. 12750', 'C.A. 24580', 244270,
239856, 349912, 342826, 4138, 330935, 6563, 349228, 350036, 24160,
17474, 349256, 2672, 113800, 248731, 363592, 35852, 348121,
'PC 17475', 36864, 350025, 223596, 'PC 17476', 'PC 17482', 113028,
7545, 250647, 348124, 34218, 36568, 347062, 350048, 12233, 250643,
113806, 315094, 36866, 236853, 'STON/O2. 3101271', 239855, 28425,
233639, 349201, 349218, 16988, 376566, 'STON/O 2. 3101288', 250648,
113773, 335097, 29103, 392096, 345780, 349204, 350042, 29108,
363294, 'SOTON/O2 3101272', 2663, 347074, 112379, 364850, 8471,
345781, 350047, 'S.O./P.P. 3', 2674, 29105, 347078, 383121, 36865,
2687, 113501, 'W./C. 6607', 'SOTON/O.Q. 3101312', 374887, 3101265,
12460, 'PC 17600', 349203, 28213, 17465, 349244, 2685, 2625,
347089, 347063, 112050, 347087, 248723, 3474, 28206, 364499,
112058, 'STON/O2. 3101290', 'S.C./PARIS 2079', 'C 7075', 315098,
19972, 368323, 367228, 2671, 347468, 2223, 'PC 17756', 315097,
392092, 11774, 'SOTON/O2 3101287', 2683, 315090, 'C.A. 5547',
349213, 347060, 'PC 17592', 392091, 113055, 2629, 350026, 28134,
17466, 233866, 236852, 'SC/PARIS 2149', 'PC 17590', 345777, 349248,
695, 345765, 2667, 349212, 349217, 349257, 7552,
'C.A./SOTON 34068', 'SOTON/OQ 392076', 211536, 112053, 111369,
370376], dtype=object)
```

Exploratory Data Analysis

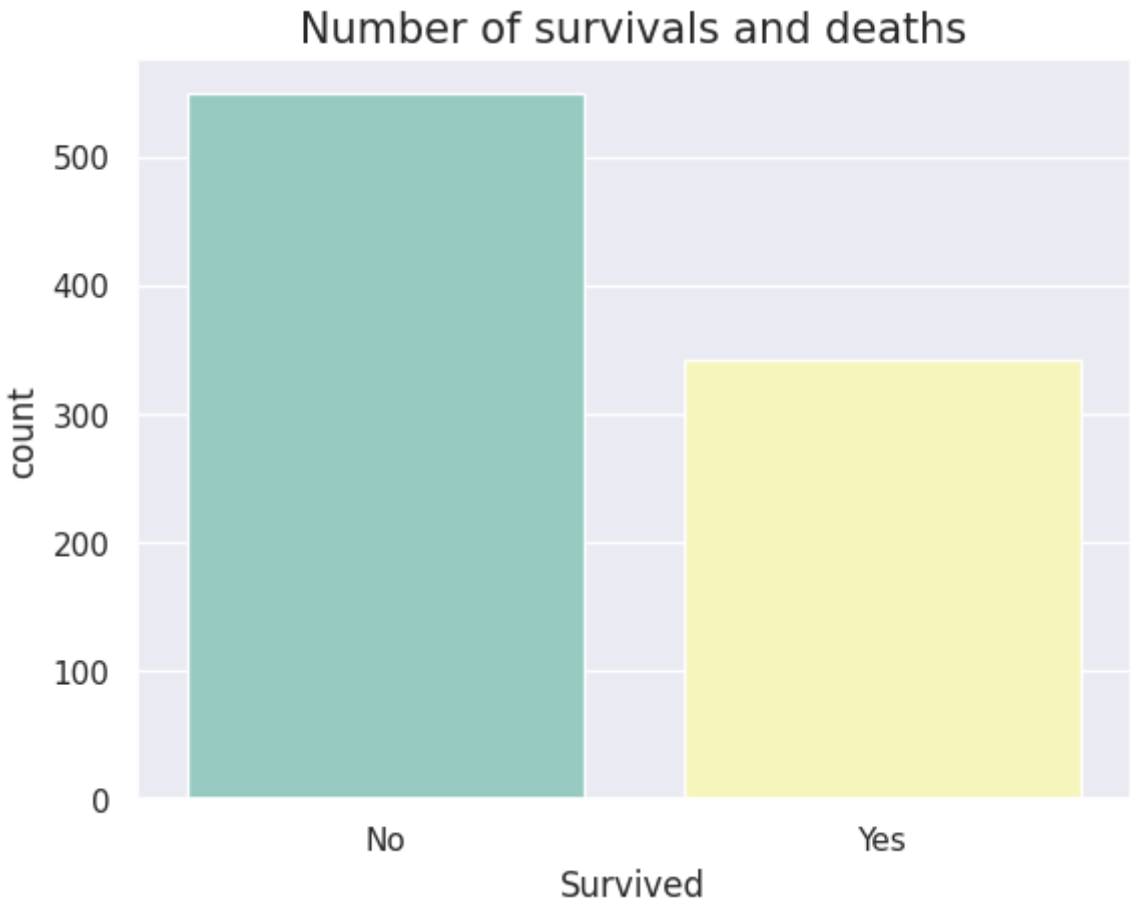
In [ ]:
```
#Dropping ID and Name columns and Ticket
ID=Train.pop('PassengerId')
Name=Train.pop('Name')
Ticket=Train.pop('Ticket')
```

In [ ]:
```
#Plotting the Numerical columns in pair plot for overall understanding
sns.set()
sns.pairplot(Train,hue='Survived')
plt.show()
```



In [ ]:
```
#Generating a categorical column of survival status for visualization and plotting
Train['Survival']=['Yes' if i==1 else 'No' for i in Train['Survived']]
```

In [ ]:
```
sns.set()
plt.title('Number of survivals and deaths',fontsize=15)
sns.countplot(x='Survived',data=Train,palette='Set3')
plt.xticks(ticks=[0,1],labels=['No','Yes'])
plt.show()
```



In [ ]:
```
Train['Survived'].value_counts()
```

Out[ ]:
```
0    549
1    342
Name: Survived, dtype: int64
```

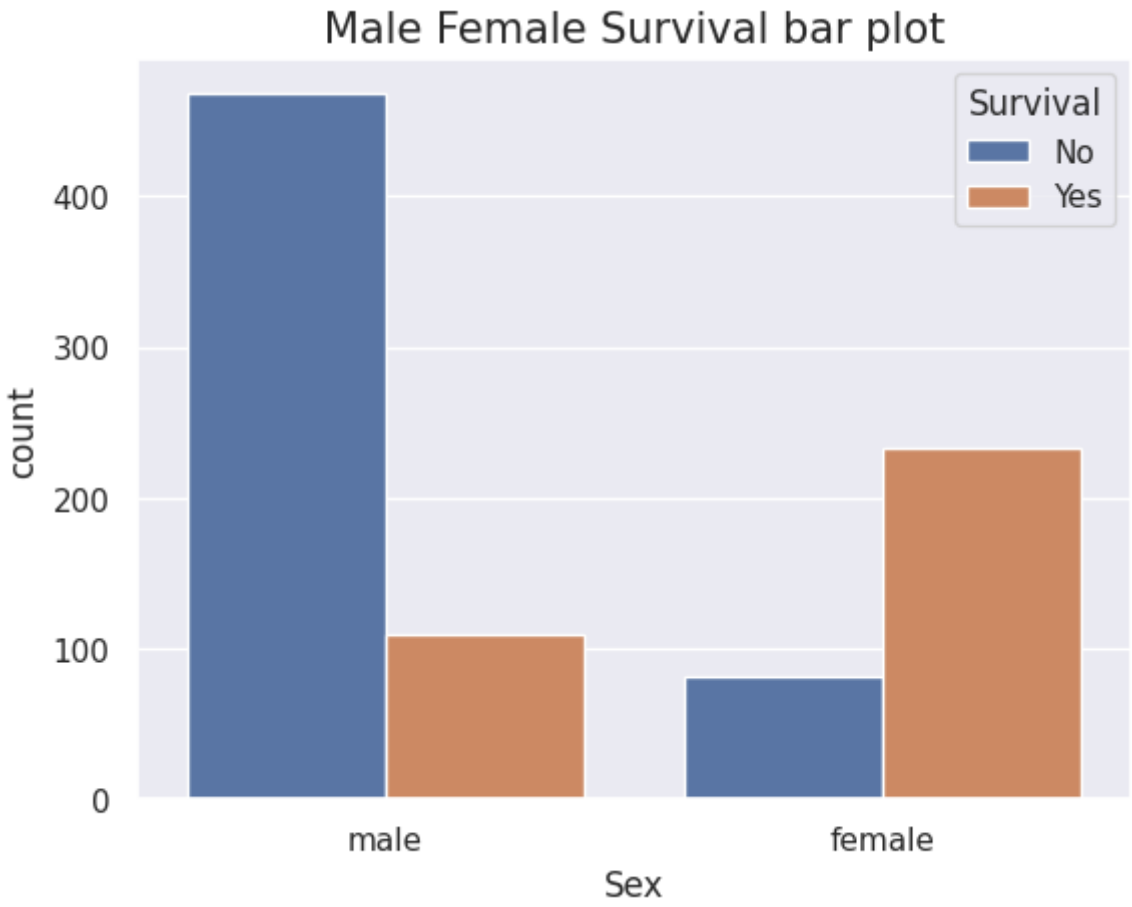
From the above graph it is clear that not many persons survived. Out of 891 persons in training dataset only 342, 38.4% of total training dataset survived.

In [ ]:
```
Train.groupby(['Sex', 'Survived'])['Survived'].count()
```

Out[ ]:
```
Sex     Survived
female  0            81
        1           233
male    0           468
        1           109
Name: Survived, dtype: int64
```

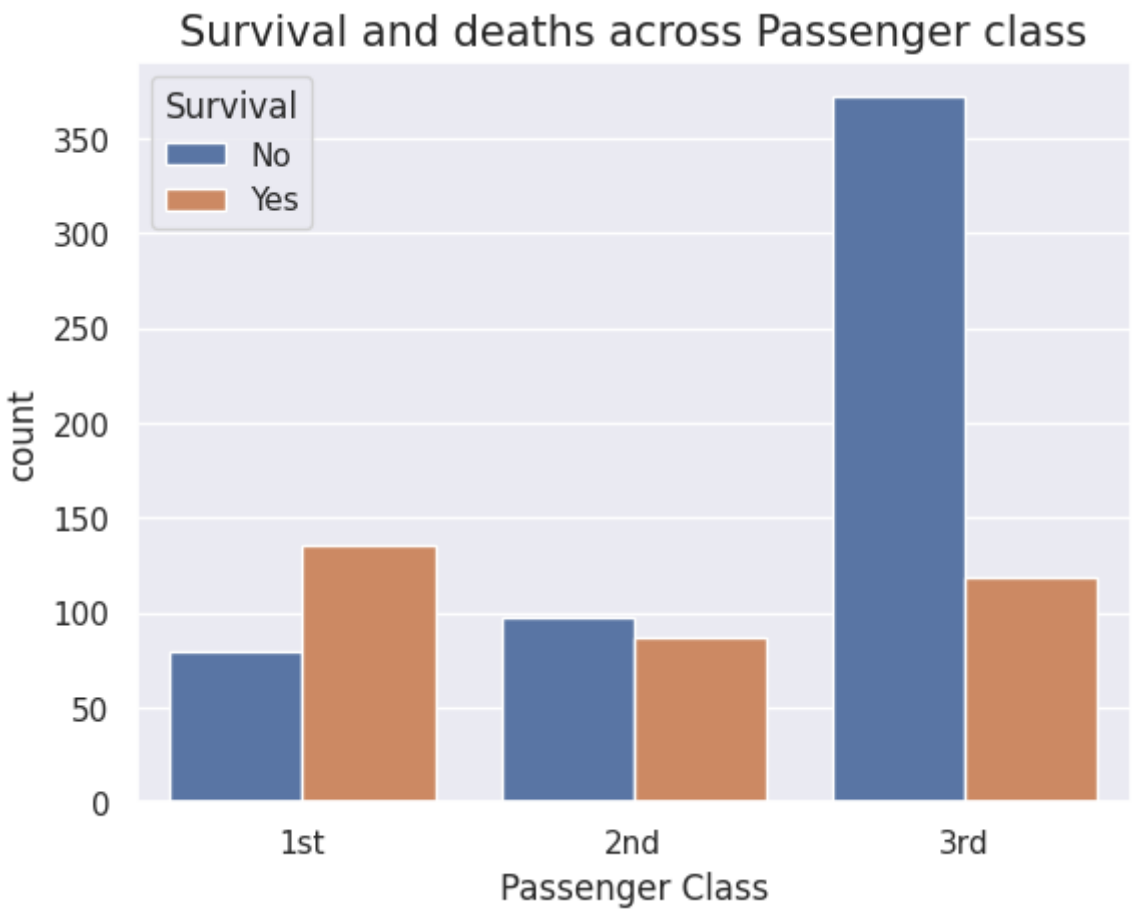
It is clear that 233 female survived out of 344. And out of 577 male 109 survived. The survival ratio of female is much greater than that of male. It can be seen clearly in following graph

In [ ]:
```
sns.set()
plt.title('Male Female Survival bar plot',fontsize=15)
sns.countplot(x='Sex',hue='Survival',data=Train)
plt.show()
```



In [ ]:

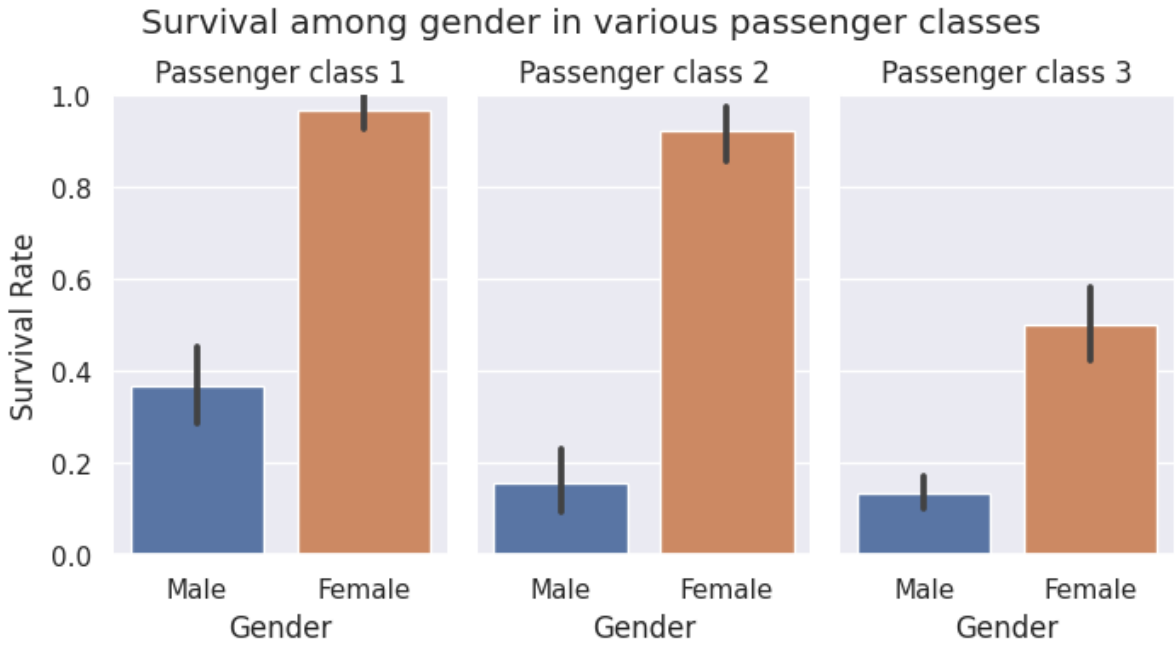
```
sns.set()
plt.title('Survival and deaths across Passenger class',fontsize=15)
sns.countplot(x='Pclass', hue='Survival', data=Train)
plt.xticks(ticks=[0,1,2],labels=['1st','2nd','3rd'])
plt.xlabel('Passenger Class')
plt.show()
```



The Death is highest among the Class 3 passengers. So simply those who paid highest were given highest priortiy at the time of survival

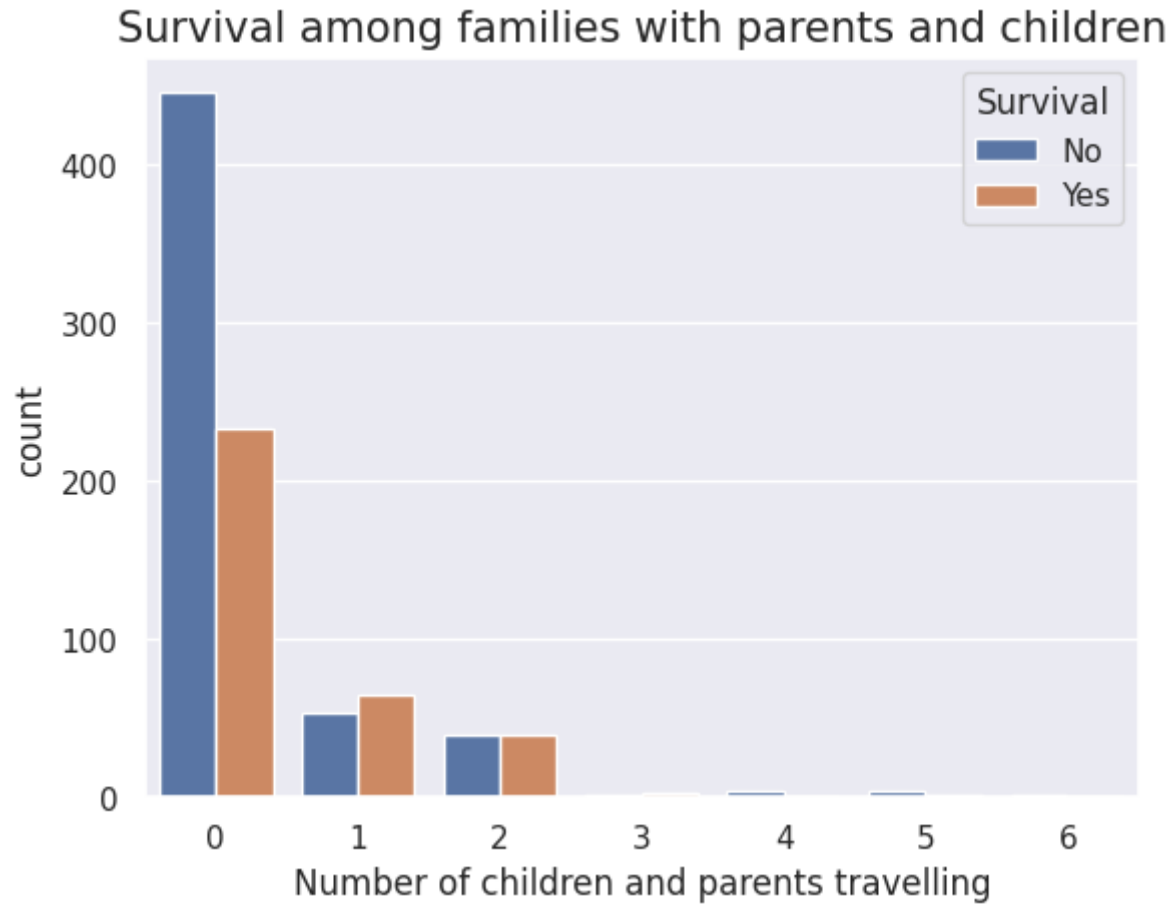
In [ ]:
```
#Combining the above two plots in single cat plot
sns.set()
plt.figure(figsize=(10,8))
g = sns.catplot(
    data=Train, x="Sex", y="Survived", col="Pclass",
    kind="bar", height=4, aspect=.6,
)
g.fig.subplots_adjust(top=0.85)
g.fig.suptitle("Survival among gender in various passenger classes")
g.set_axis_labels("Gender", "Survival Rate")
g.set_xticklabels(["Male", "Female"])
g.set_titles("Passenger class {col_name}")
g.set(ylim=(0, 1))
g.despine(left=True)
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7ea3e65e7430>

<Figure size 1000x800 with 0 Axes>
```

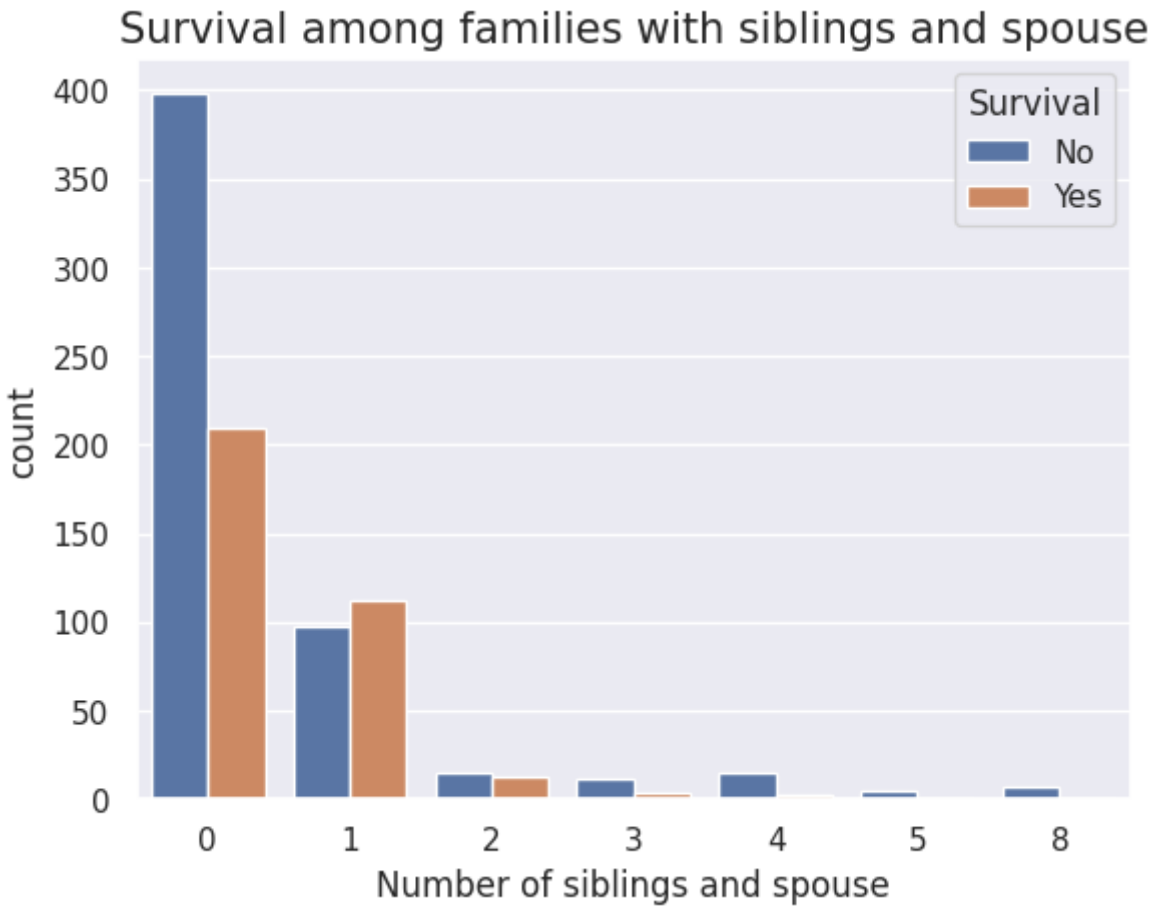


So in every classes Females survived more

In [ ]:
```
sns.set()
plt.title('Survival among families with parents and children',fontsize=15)
sns.countplot(x='Parch',hue='Survival',data=Train)
plt.xlabel('Number of children and parents travelling')
plt.show()
```



In [ ]:
```
plt.title('Survival among families with siblings and spouse',fontsize=15)
sns.countplot(x='SibSp',hue='Survival',data=Train)
plt.xlabel('Number of siblings and spouse')
plt.show()
```



We see survival rate is comparatively higher for those who are travelling with families than alone

In [ ]:
```
#We can merge the SibSp and Parch column to single column as Number of Family membe
Train['Number of Family members']=Train['Parch']+Train['SibSp']
Train[['Number of Family members','Parch','SibSp']]
```

Out[ ]:

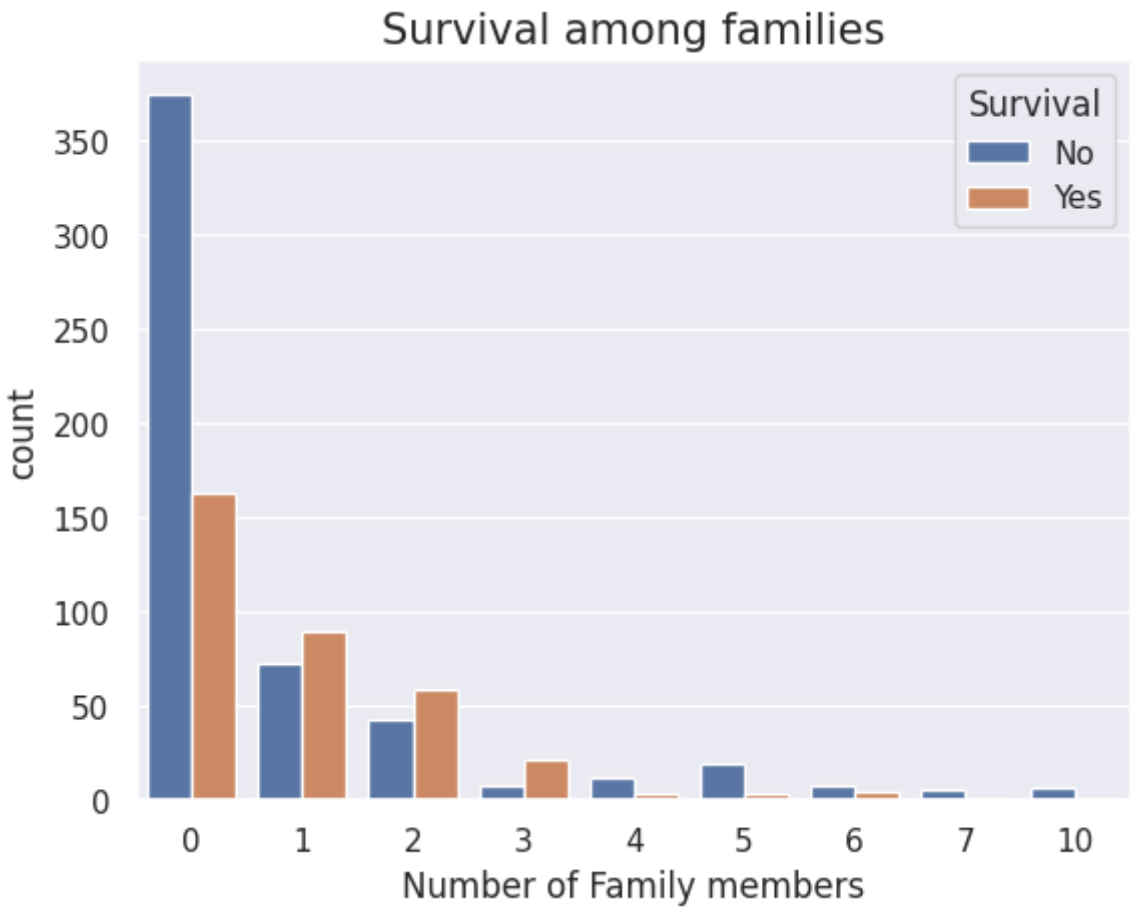
	Number of Family members	Parch	SibSp
0	1	0	1
1	1	0	1
2	0	0	0
3	1	0	1
4	0	0	0
...	...	...	...
886	0	0	0
887	0	0	0
888	3	2	1
889	0	0	0
890	0	0	0

891 rows × 3 columns

In [ ]:
```
#Dropping Parch and SibSp
Train.drop('Parch',axis=1,inplace=True)
```

```
Train.drop('SibSp',axis=1,inplace=True)
```

In [ ]:
```
plt.title('Survival among families',fontsize=15)
sns.countplot(x='Number of Family members',hue='Survival',data=Train)
plt.show()
```

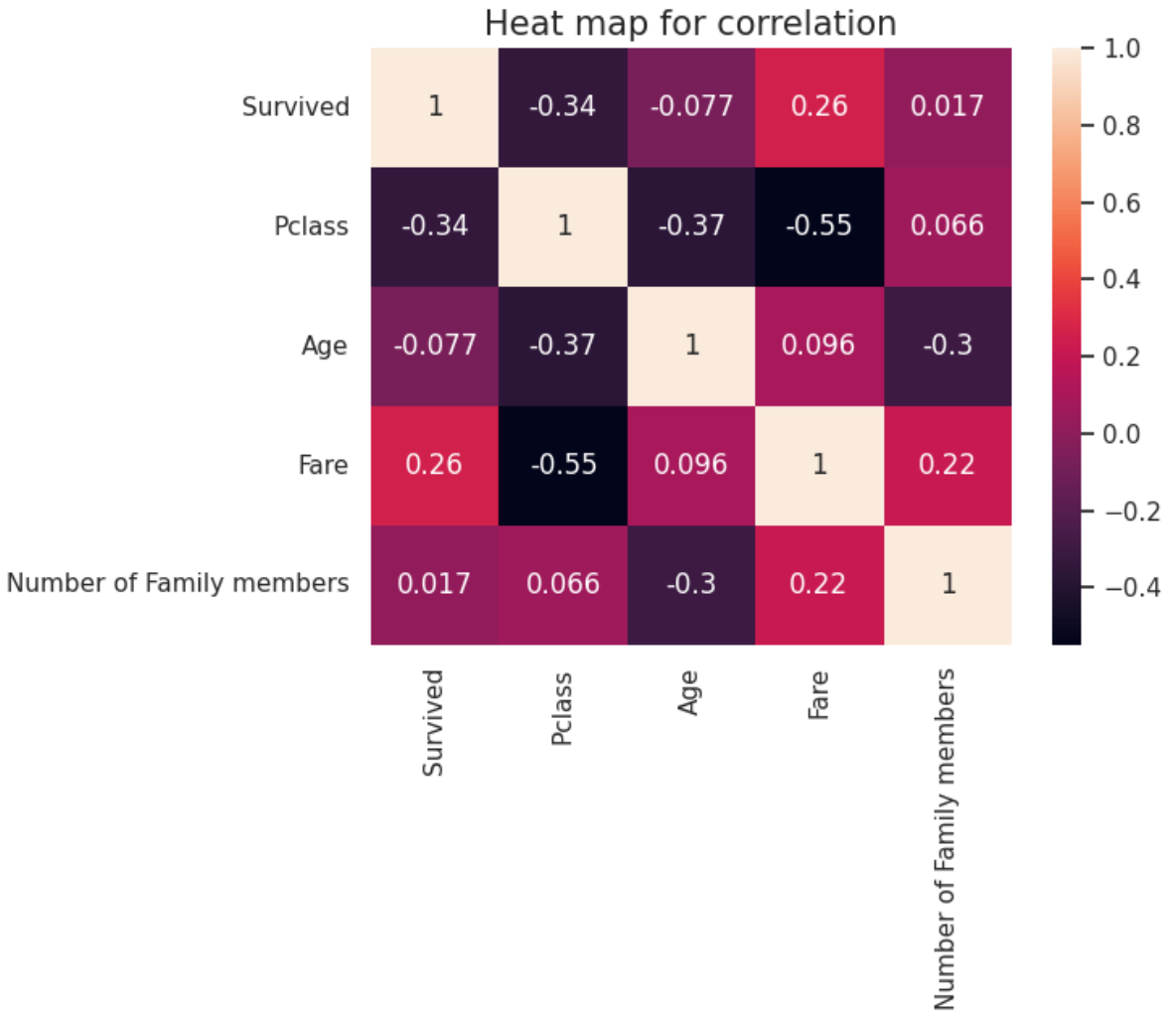


In [ ]:
```
Train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Survived                 891 non-null    int64
 1   Pclass                   891 non-null    int64
 2   Sex                      891 non-null    object
 3   Age                      714 non-null    float64
 4   Fare                     891 non-null    float64
 5   Cabin                    204 non-null    object
 6   Embarked                 889 non-null    object
 7   Survival                 891 non-null    object
 8   Number of Family members  891 non-null    int64
dtypes: float64(2), int64(3), object(4)
memory usage: 62.8+ KB
```

In [ ]:
```
#Correlation plot
Cor=Train.corr()
sns.heatmap(Cor,annot=True)
plt.title("Heat map for correlation",fontsize=15)
plt.show()
```

```
<ipython-input-30-473a0c915f40>:2: FutureWarning: The default value of numeric_onl
y in DataFrame.corr is deprecated. In a future version, it will default to False.
Select only valid columns or specify the value of numeric_only to silence this war
ning.
  Cor=Train.corr()
```

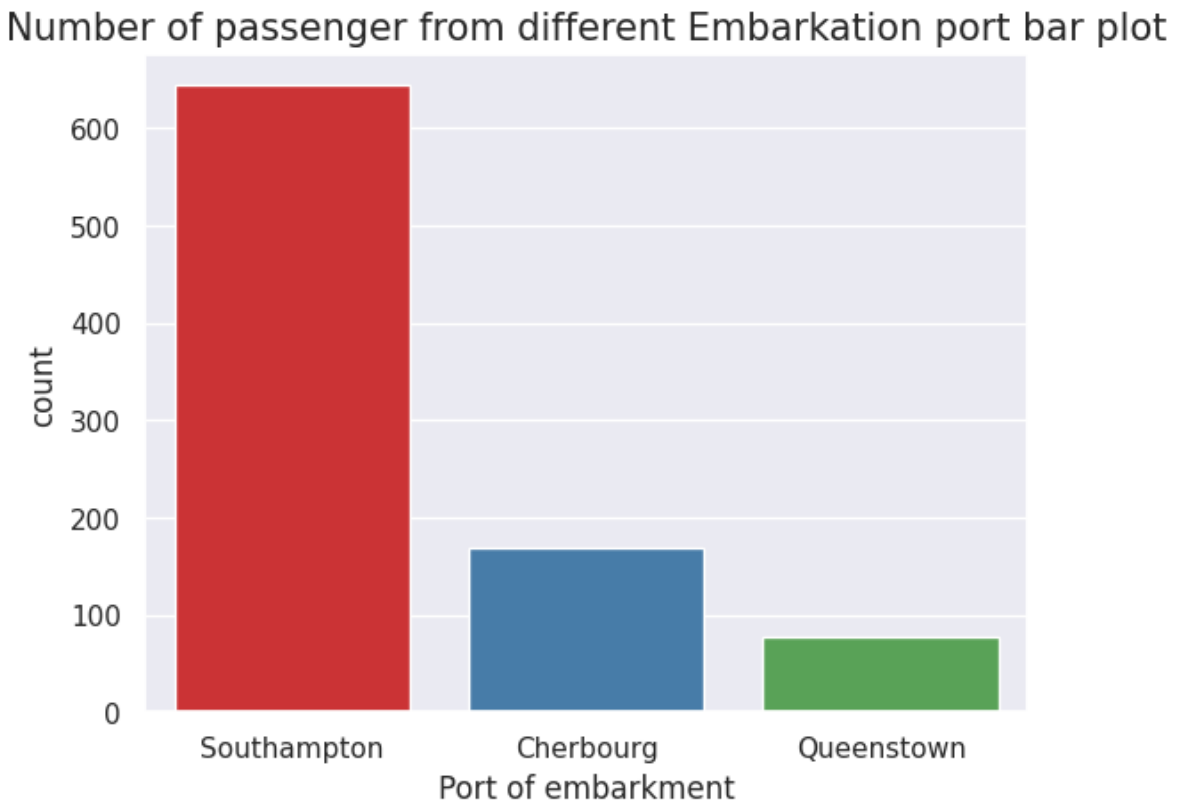


In [ ]:
```
#The numerical features are less correlated
```

Handling Missing Values

Embarked Column

In [ ]:
```
sns.set()
labels=['Southampton','Cherbourg','Queenstown']
plt.title('Number of passenger from different Embarkation port bar plot',fontsize=1
sns.countplot(x='Embarked', data=Train, palette='Set1')
plt.xticks(ticks=[0,1,2],labels=labels)
plt.xlabel("Port of embarkment")
plt.show()
```

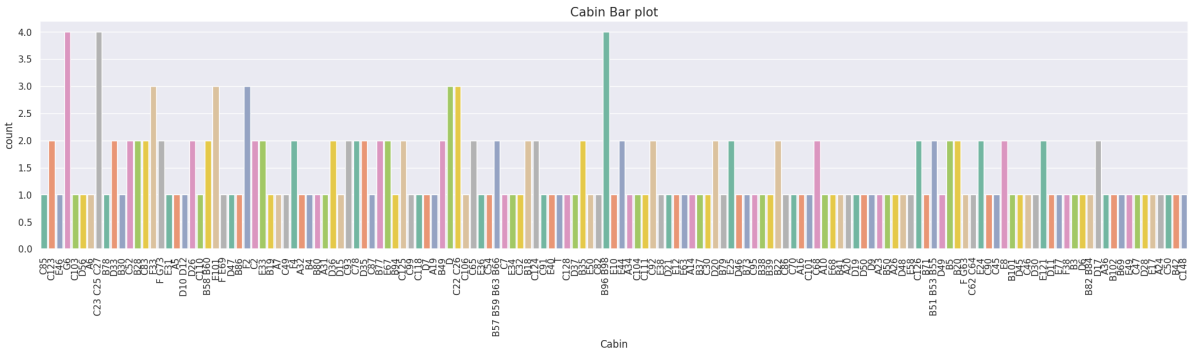


We see Most of the pax boarded from S (=Southampton) > C (=Cherbourg) > Q (=Queenstown). So it make sense to replace the missing values with S

In [ ]:
```
Train['Embarked'].fillna('S',inplace=True)
```

Cabin

In [ ]:
```
plt.figure(figsize=(25,5))
plt.title('Cabin Bar plot',fontsize=15)
sns.countplot(x='Cabin',data=Train,palette='Set2')
plt.xticks(rotation=90)
plt.show()
```



In [ ]:
```
Train[['Survived','Cabin']]
```

Out[ ]:

	Survived	Cabin
0	0	NaN
1	1	C85
2	1	NaN
3	1	C123
4	0	NaN
...	...	...
886	0	NaN
887	1	B42
888	0	NaN
889	1	C148
890	0	NaN

891 rows × 2 columns

In [ ]:
```
print('Number of missing entries in Cabin :',Train['Cabin'].isna().sum())
#Looking at the distribution and number of missing entries we can state that there
#information as such can be retrieved from this column. Also becasue it is categori
#So dropping this column is better
Train.drop('Cabin',axis=1,inplace=True)
```

```
Number of missing entries in Cabin : 687
```

In [ ]:
```
Train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Survived                  891 non-null    int64  
 1   Pclass                    891 non-null    int64  
 2   Sex                       891 non-null    object 
 3   Age                       714 non-null    float64
 4   Fare                      891 non-null    float64
 5   Embarked                  891 non-null    object 
 6   Survival                  891 non-null    object 
 7   Number of Family members  891 non-null    int64  
dtypes: float64(2), int64(3), object(3)
memory usage: 55.8+ KB
```

In [ ]:
```
#We will one hot encode Pclass, Sex, Embarked
Embarked=Train['Embarked']
One_h_Train=pd.get_dummies(Train, columns=["Pclass","Embarked","Sex"])
One_h_Train.drop('Sex_female', axis=1, inplace=True)
```

In [ ]:
```
One_h_Train.head()
```

Out[ ]:

	Survived	Age	Fare	Survival	Number of Family members	Pclass_1	Pclass_2	Pclass_3	Embarked_C	Embarke
0	0	22.0	7.2500	No	1	0	0	1	0	
1	1	38.0	71.2833	Yes	1	1	0	0	1	
2	1	26.0	7.9250	Yes	0	0	0	1	0	
3	1	35.0	53.1000	Yes	1	1	0	0	0	
4	0	35.0	8.0500	No	0	0	0	1	0	

In [ ]:
```
Oh_rep=pd.DataFrame({"Emabarked":Embarked,"Embarked_C":One_h_Train["Embarked_C"],"E
```

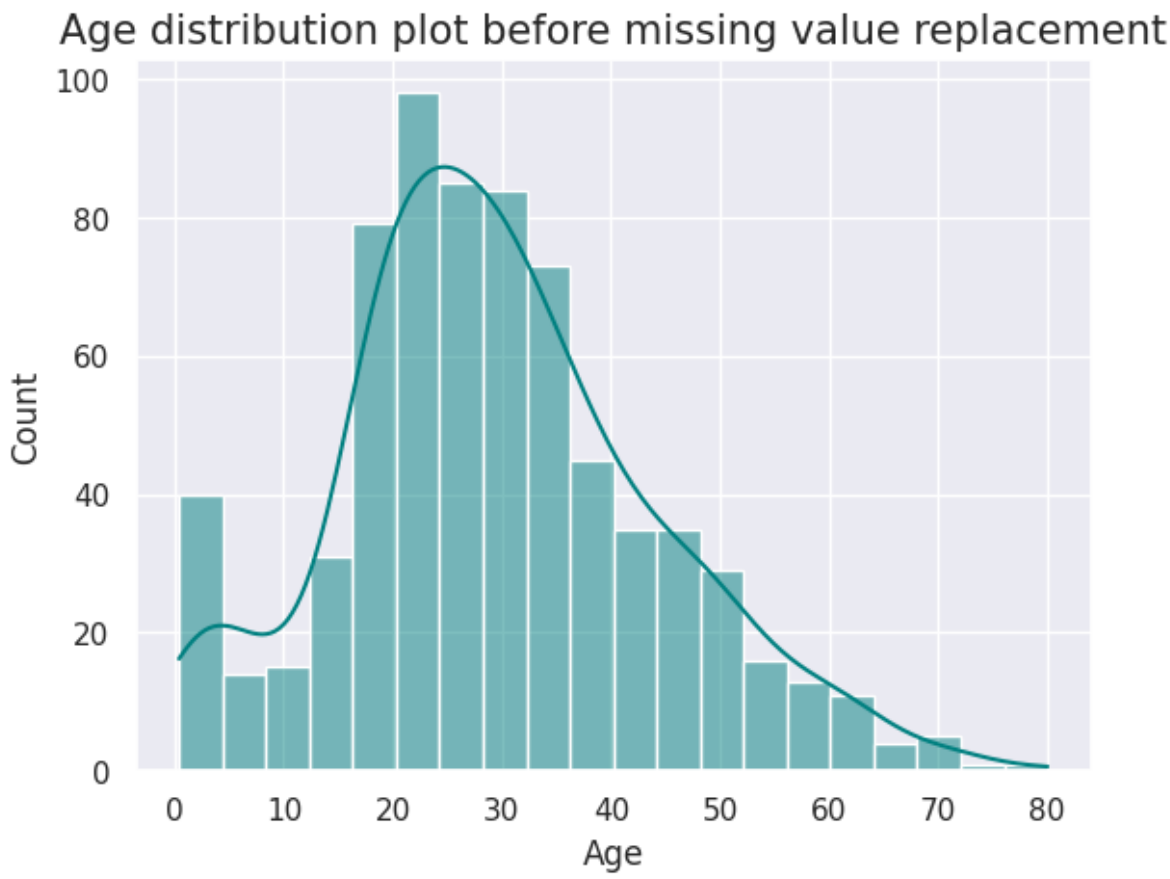
In [ ]:
```
Data=Oh_rep.head()
Data.to_csv("Data.csv")
```

In [ ]:
```
Target=One_h_Train.pop('Survived')
One_h_Train.drop('Survival',axis=1,inplace=True)
```

Age column

With skewed data median imputation is good choice. Mean imputation works well with near normal distributions. For Categorical columns Mode Imputation method is the appropriate method. But here we will use KNN for imputation purpose

In [ ]:
```
sns.set()
sns.histplot(data=One_h_Train,x='Age',kde=True,color='teal')
plt.title("Age distribution plot before missing value replacement",fontsize=15)
plt.show()
```



In [ ]:
```
cols=One_h_Train.columns
age_imputer=KNNImputer(n_neighbors=3)
Imputer_age=age_imputer.fit_transform(One_h_Train)
```

In [ ]:
```
One_h_Train['Age']=Imputer_age[:,0]
```

In [ ]:
```
merged_data=pd.concat([One_h_Train,Train['Survival']],axis=1)
```

In [ ]:
```
merged_data
```

Out[ ]:

	Age	Fare	Number of Family members	Pclass_1	Pclass_2	Pclass_3	Embarked_C	Embarked_Q	Embark
0	22.000000	7.2500	1	0	0	1	0	0	
1	38.000000	71.2833	1	1	0	0	1	0	
2	26.000000	7.9250	0	0	0	1	0	0	
3	35.000000	53.1000	1	1	0	0	0	0	
4	35.000000	8.0500	0	0	0	1	0	0	
...	...	...	...	...	...	...	...	...	
886	27.000000	13.0000	0	0	1	0	0	0	
887	19.000000	30.0000	0	1	0	0	0	0	
888	21.333333	23.4500	3	0	0	1	0	0	
889	26.000000	30.0000	0	1	0	0	1	0	
890	32.000000	7.7500	0	0	0	1	0	1	

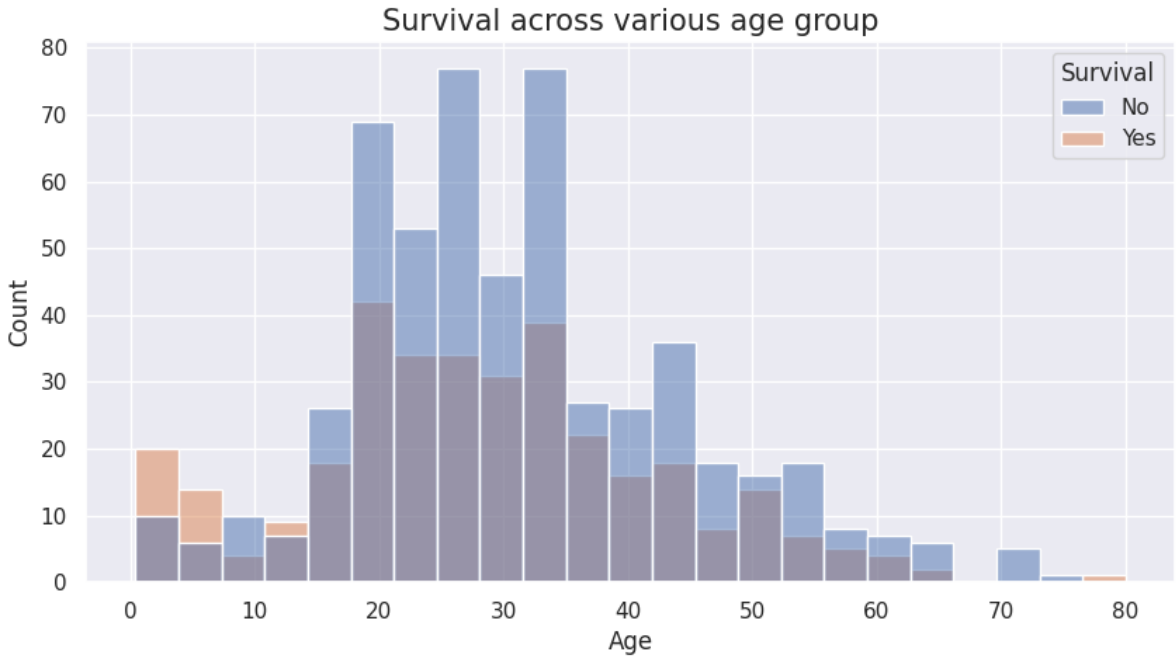
891 rows × 11 columns

In [ ]:
```
One_h_Train.describe()
```

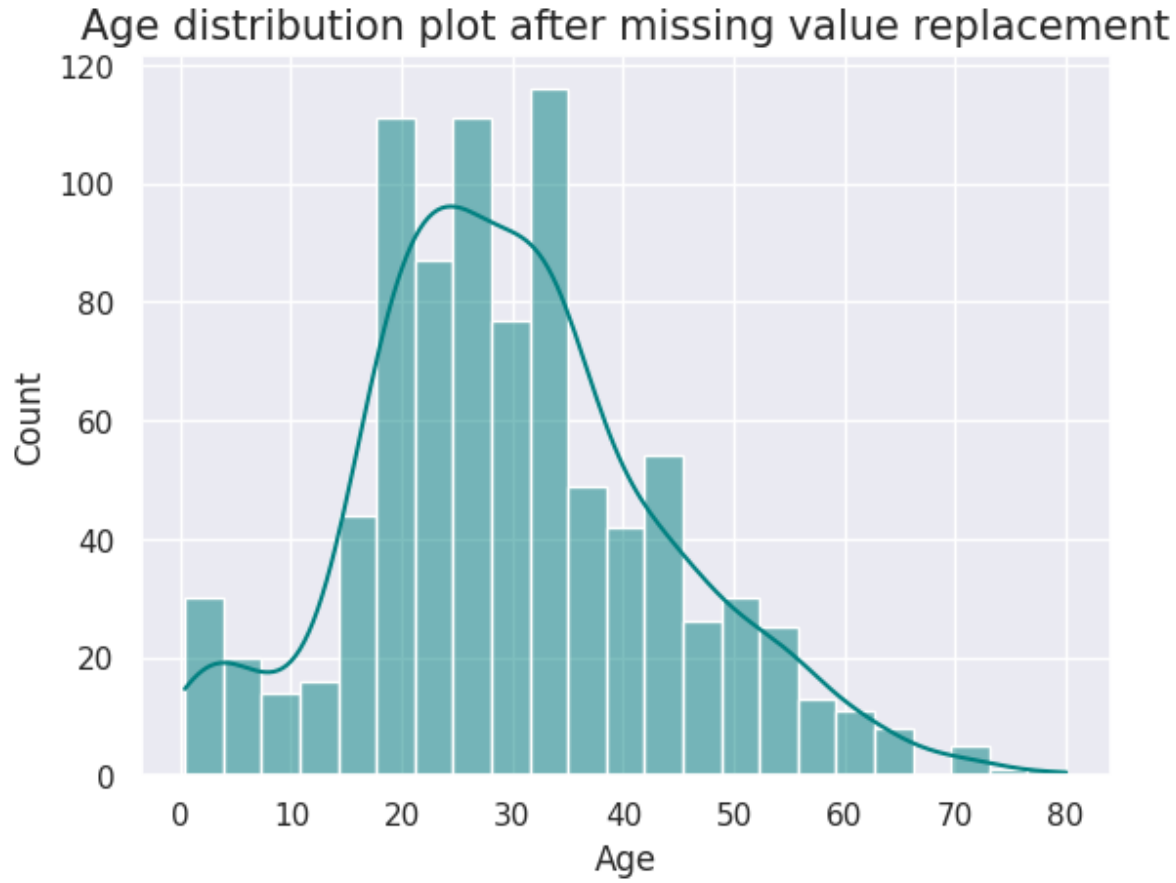
Out[ ]:

	Age	Fare	Number of Family members	Pclass_1	Pclass_2	Pclass_3	Embarked_C	Emba
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891
mean	30.057243	32.204208	0.904602	0.242424	0.206510	0.551066	0.188552	0
std	13.834869	49.693429	1.613459	0.428790	0.405028	0.497665	0.391372	0
min	0.420000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	21.000000	7.910400	0.000000	0.000000	0.000000	0.000000	0.000000	0
50%	29.000000	14.454200	0.000000	0.000000	0.000000	1.000000	0.000000	0
75%	38.000000	31.000000	1.000000	0.000000	0.000000	1.000000	0.000000	0
max	80.000000	512.329200	10.000000	1.000000	1.000000	1.000000	1.000000	1

In [ ]:
```
plt.figure(figsize=(10,5))
plt.title("Survival across various age group",fontsize=15)
ax=sns.histplot(x='Age',hue='Survival',data=merged_data)
plt.show()
```



In [ ]:
```
sns.set()
sns.histplot(data=One_h_Train,x='Age',kde=True,color='teal')
plt.title("Age distribution plot after missing value replacement",fontsize=15)
plt.show()
```



In [ ]:
```
One_h_Train.info()
```

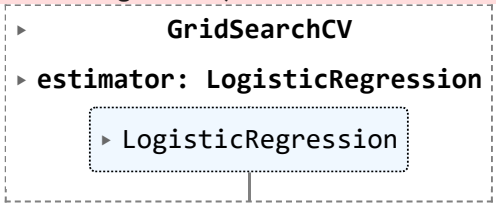
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       891 non-null    float64
 1   Fare                      891 non-null    float64
 2   Number of Family members  891 non-null    int64
 3   Pclass_1                  891 non-null    uint8
 4   Pclass_2                  891 non-null    uint8
 5   Pclass_3                  891 non-null    uint8
 6   Embarked_C                891 non-null    uint8
 7   Embarked_Q                891 non-null    uint8
 8   Embarked_S                891 non-null    uint8
 9   Sex_male                  891 non-null    uint8
dtypes: float64(2), int64(1), uint8(7)
memory usage: 27.1 KB
```

In [ ]:
```
X_train,X_val,y_train,y_val=train_test_split(One_h_Train,Target,random_state=20,tes
S=MinMaxScaler()
X_train_sc=S.fit_transform(X_train)#Scale the data between 0 and 1
X_val_sc=S.transform(X_val)
```

In [ ]:
```
logreg=LogisticRegression()
grid={"penalty":["l1","l2"],"max_iter":[50,100,1000,20000,400000]}
logreg_cv=GridSearchCV(logreg,grid,cv=10)
logreg_cv.fit(X_train_sc,y_train)# Fit the model
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:37
8: FitFailedWarning:
50 fits failed out of a total of 100.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_sco
re='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
50 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validatio
n.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.p
y", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.p
y", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:952: Us
erWarning: One or more of the test scores are non-finite: [       nan 0.78791603
nan 0.78791603        nan 0.78791603
        nan 0.78791603        nan 0.78791603]
  warnings.warn(
```

Out[ ]:



In [ ]:
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)# Get the estimator
```

```
tuned hpyerparameters :(best parameters)  {'max_iter': 50, 'penalty': 'l2'}
accuracy : 0.7879160266257039
```

In [ ]:
```
print("Accuracy on Train set :",logreg_cv.best_estimator_.score(X_train_sc,y_train)
```

```
Accuracy on Train set : 0.7945425361155698
```

In [ ]:
```
print("Accuracy on Validation set :",logreg_cv.best_estimator_.score(X_val_sc,y_val
```

```
Accuracy on Validation set : 0.8283582089552238
```

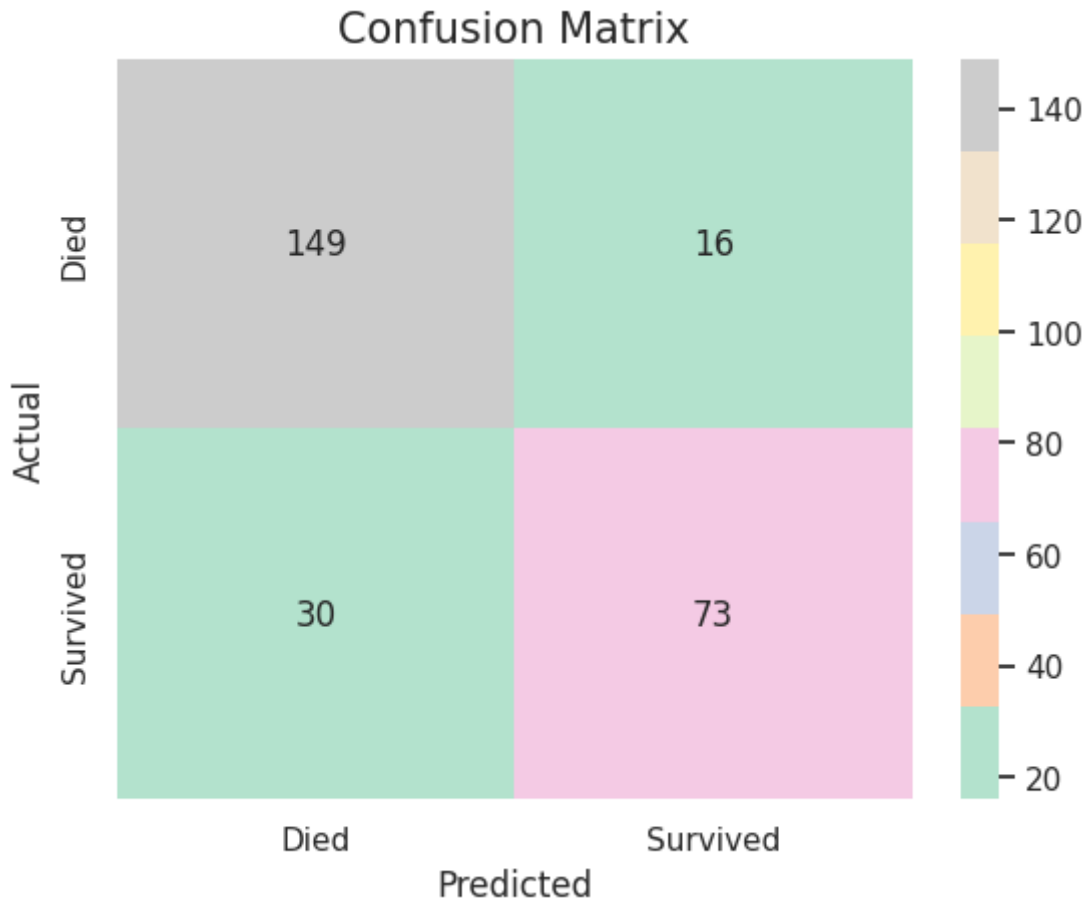
In [ ]:
```
yhat_val=logreg_cv.best_estimator_.predict(X_val_sc)#Predict for the validation set
```

Model evaluation and Result visualization

In [ ]:
```
print(classification_report(y_val,yhat_val))
```

```
              precision    recall  f1-score   support

           0       0.83      0.90      0.87       165
           1       0.82      0.71      0.76       103

    accuracy                           0.83       268
   macro avg       0.83      0.81      0.81       268
weighted avg       0.83      0.83      0.83       268
```

In [ ]:
```
#Get the confusion matrix on validation set
cm=confusion_matrix(y_val,yhat_val)
plt.title("Confusion Matrix",fontsize=15)
sns.heatmap(cm,annot=True,cmap='Pastel2',fmt='d')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.xticks(ticks=[0.5,1.5],labels=['Died','Survived'])
plt.yticks(ticks=[0.5,1.5],labels=['Died','Survived'])
plt.show()
```



True Positive=48 (Actual survived, predicted survived)

False Positive=21 (Actual died, predicted survived)

False Negetive=12 (Actual survived, predicted died)

True Negative=98 (Actual died, predicted died)

Checking ROC

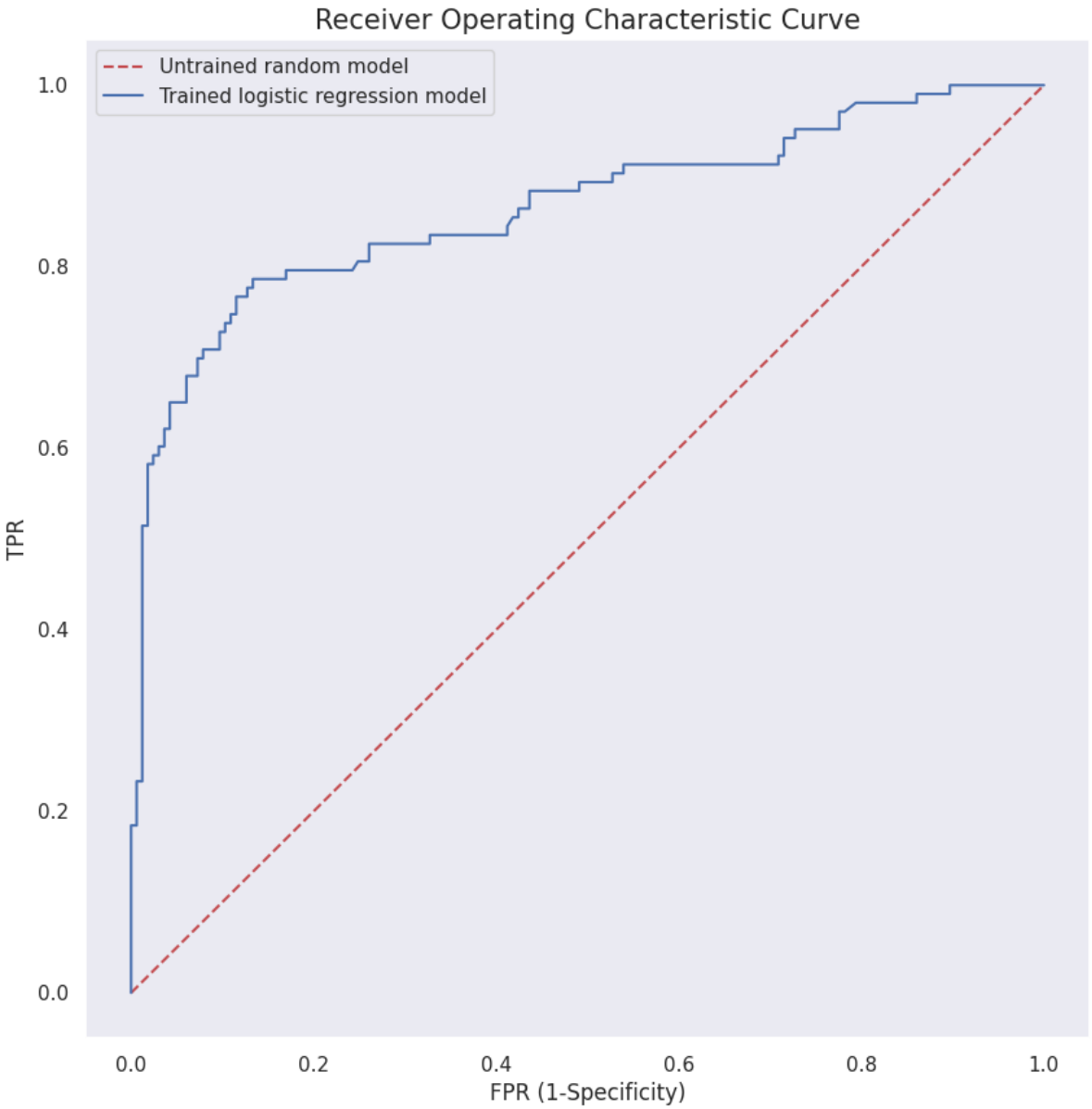
In [ ]:
```
yhat_val_prob=logreg_cv.best_estimator_.predict_proba(X_val_sc)[:,1]
fpr,tpr,threshold=roc_curve(y_val,yhat_val_prob)
print("Area of ROC is",auc(fpr,tpr))
```

```
Area of ROC is 0.8643130332450721
```

An AU ROC more than 0.5 indicates a working model. Above 0.9 is an excellent model. for between 0.8 and 0.9 is considered a good model. Below 0.7 is inferior. So our model is fair enough

In [ ]:
```
plt.figure(figsize=(10,10))
plt.plot([0,1],[0,1],'r--',label='Untrained random model')
plt.plot(fpr,tpr,label='Trained logistic regression model')
plt.title("Receiver Operating Characteristic Curve",fontsize=15)
plt.ylabel("TPR")
```

```
plt.xlabel("FPR (1-Specificity)")
plt.legend()
plt.grid()
plt.show()
```



Get the model weights

In [ ]:
```
print("Weights are : ",logreg_cv.best_estimator_.coef_[0])
print("bias is : ",logreg_cv.best_estimator_.intercept_)
```

```
Weights are :  [-1.72947805  0.21167606 -1.34677625  0.9596724   0.22759791 -1.186
88561
  0.19601209  0.15405493 -0.34968233 -2.35825682]
bias is :  [2.30207894]
```

Visualization

In [ ]:
```
Test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

In [ ]:
```
#Dropping Cabin column in the test set
Test.drop('Cabin',axis=1,inplace=True)
Test.drop('Ticket',axis=1,inplace=True)
Test['Number of Family members']=Test['Parch']+Test['SibSp']# add the correlated 'P
Test.drop('Parch',axis=1,inplace=True)
Test.drop('SibSp',axis=1,inplace=True)
ID_test=Test.pop('PassengerId')
Name_test=Test.pop('Name')
Test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 6 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Pclass                   418 non-null    int64  
 1   Sex                      418 non-null    object 
 2   Age                      332 non-null    float64
 3   Fare                     417 non-null    float64
 4   Embarked                 418 non-null    object 
 5   Number of Family members  418 non-null    int64  
dtypes: float64(2), int64(2), object(2)
memory usage: 19.7+ KB
```

In [ ]:
```
#Converting to one hot encoding
One_h_Test=pd.get_dummies(Test, columns=["Pclass","Embarked","Sex"])
One_h_Test.drop('Sex_female', axis=1, inplace=True)
```

In [ ]:
```
One_h_Test.columns
```

Out[ ]:
```
Index(['Age', 'Fare', 'Number of Family members', 'Pclass_1', 'Pclass_2',
       'Pclass_3', 'Embarked_C', 'Embarked_Q', 'Embarked_S', 'Sex_male'],
      dtype='object')
```

In [ ]:
```
imputer=KNNImputer(n_neighbors=3)
Imputer=imputer.fit_transform(One_h_Test)
One_h_Test['Age']=Imputer[:,0]
One_h_Test['Fare']=Imputer[:,1]
```

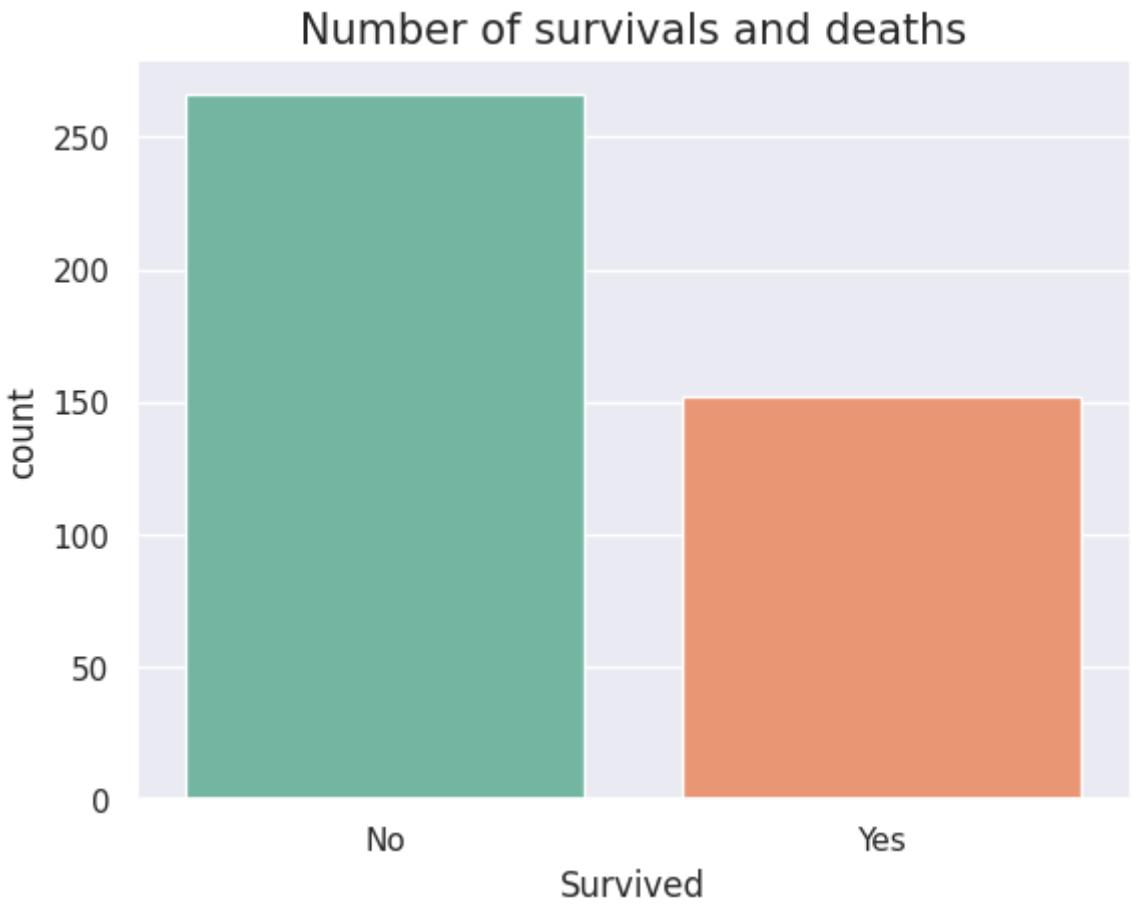
In [ ]:
```
One_h_Test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 10 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       418 non-null    float64
 1   Fare                      418 non-null    float64
 2   Number of Family members  418 non-null    int64
 3   Pclass_1                  418 non-null    uint8
 4   Pclass_2                  418 non-null    uint8
 5   Pclass_3                  418 non-null    uint8
 6   Embarked_C                418 non-null    uint8
 7   Embarked_Q                418 non-null    uint8
 8   Embarked_S                418 non-null    uint8
 9   Sex_male                  418 non-null    uint8
dtypes: float64(2), int64(1), uint8(7)
memory usage: 12.8 KB
```

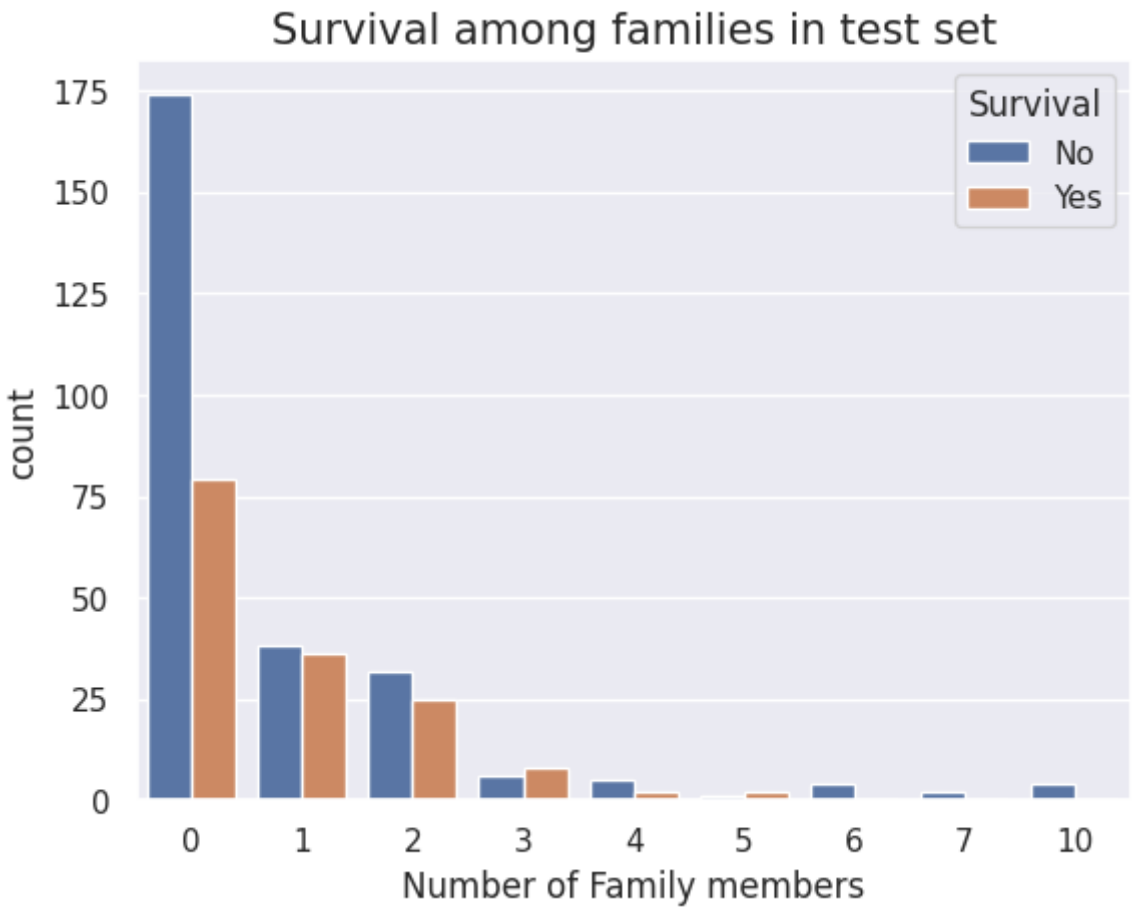
In [ ]:
```
#Scale the data
X_test_Test=S.transform(One_h_Test)
Survival=logreg_cv.best_estimator_.predict(X_test_Test)#Make prediction on the test
Test['Survived']=Survival
Test['Survival']=['Yes' if i==1 else 'No' for i in Survival]#Make a catogorical sur
Test['Survived'].value_counts()# in the test set we are finding only 9 people survi
```

Out[ ]:
```
0    266
1    152
Name: Survived, dtype: int64
```

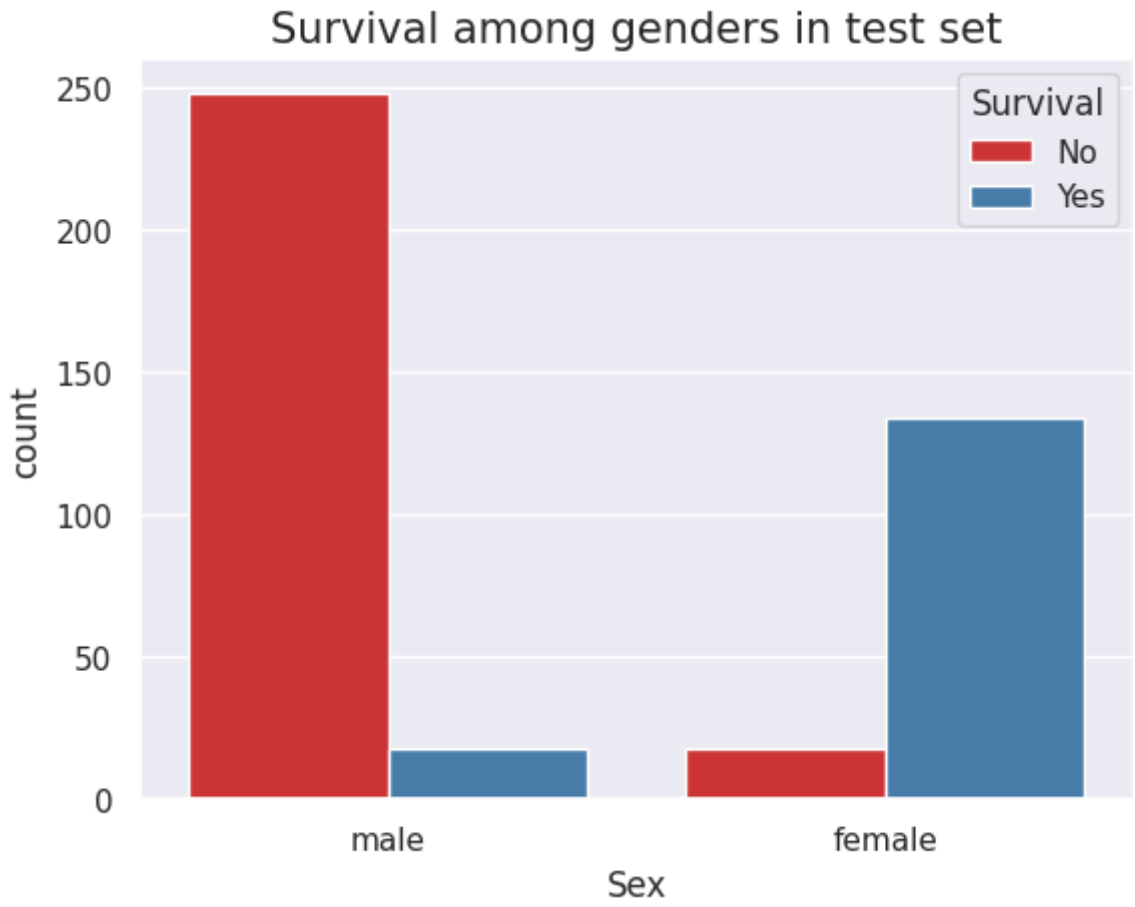
In [ ]:
```
sns.set()
plt.title('Number of survivals and deaths',fontsize=15)
sns.countplot(x='Survived',data=Test,palette='Set2')
plt.xticks(ticks=[0,1],labels=['No','Yes'])
plt.show()
```



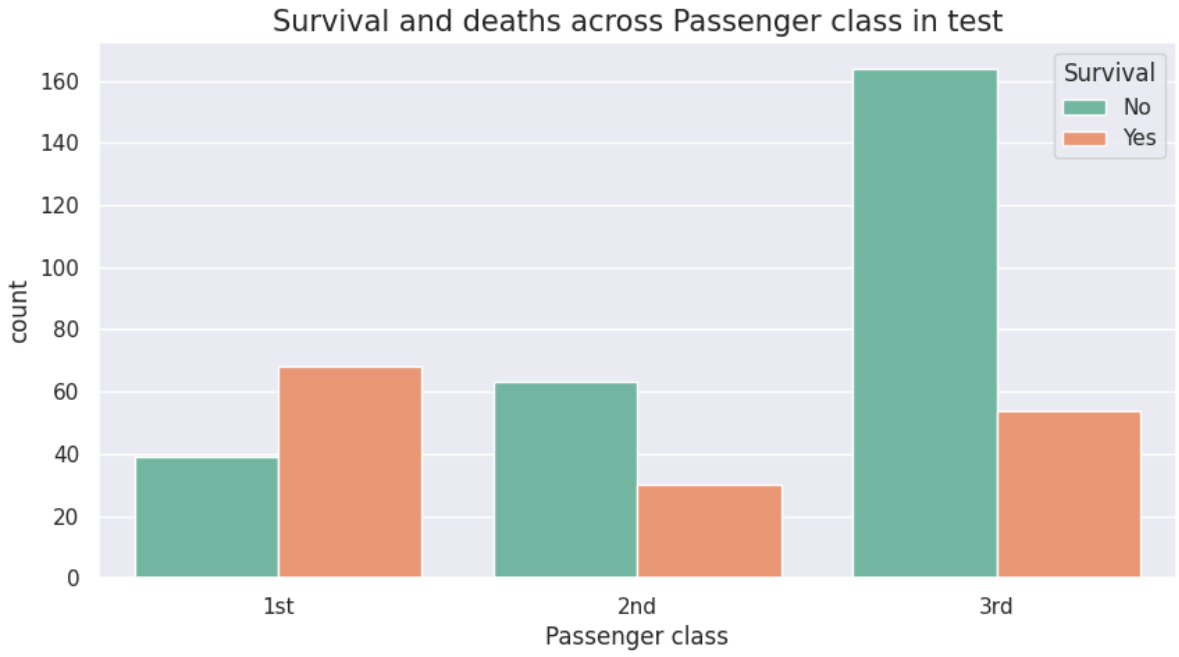
In [ ]:
```
plt.title('Survival among families in test set',fontsize=15)#Obtaining the survival
sns.countplot(x='Number of Family members',hue='Survival',data=Test)
plt.show()
```



In [ ]:
```
plt.title('Survival among genders in test set',fontsize=15)# Check the gender based
sns.countplot(x='Sex',hue='Survival',data=Test,palette='Set1')
plt.show()
```



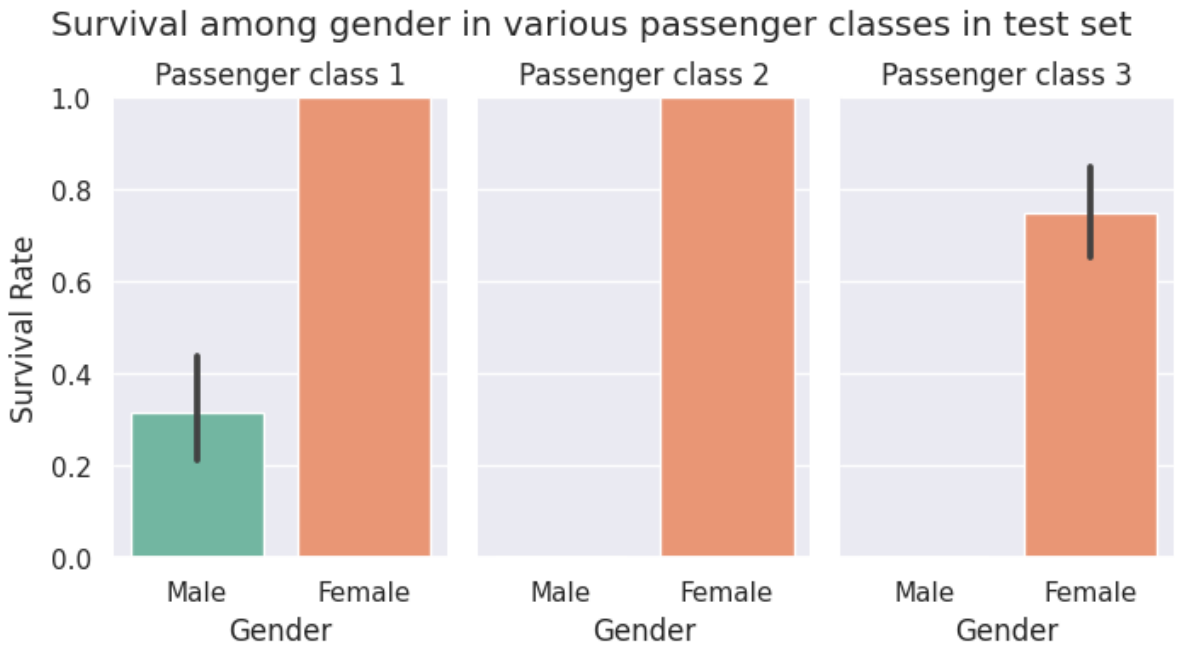
In [ ]:
```
plt.figure(figsize=(10,5))
plt.title("Survival and deaths across Passenger class in test",fontsize=15)
ax=sns.countplot(x='Pclass',hue='Survival',data=Test,palette='Set2')
plt.xticks(ticks=[0,1,2],labels=['1st','2nd','3rd'])
plt.xlabel('Passenger class')
plt.show()
```



In [ ]:
```
print("Number of survivals in 1st class ",Test[(Test['Pclass']==1) & (Test['Survive
print("Number of deaths in 1st class ",len(Test[(Test['Pclass']==1) & (Test['Surviv
print("Number of survivals in 2nd class ",Test[(Test['Pclass']==2) & (Test['Survive
print("Number of deaths in 2nd class ",len(Test[(Test['Pclass']==2) & (Test['Surviv
print("Number of survivals in 3rd class ",Test[(Test['Pclass']==3) & (Test['Survive
print("Number of deaths in 3rd class ",len(Test[(Test['Pclass']==3) & (Test['Surviv
```

```
Number of survivals in 1st class  68
Number of deaths in 1st class  39
Number of survivals in 2nd class  30
Number of deaths in 2nd class  63
Number of survivals in 3rd class  54
Number of deaths in 3rd class  164
```

In [ ]:
```
g = sns.catplot(
    data=Test, x="Sex", y="Survived", col="Pclass",
    kind="bar", height=4, aspect=.6,palette='Set2'
)
g.fig.subplots_adjust(top=0.85)
g.fig.suptitle("Survival among gender in various passenger classes in test set")
g.set_axis_labels("Gender", "Survival Rate")
g.set_xticklabels(["Male", "Female"])
g.set_titles("Passenger class {col_name}")
g.set(ylim=(0, 1))
g.despine(left=True)
plt.show()
```



In [ ]:
```
plt.figure(figsize=(10,5))
plt.title("Survival across various age group in test set",fontsize=15)
ax=sns.histplot(x='Age',hue='Survival',data=Test)
plt.show()
```


Survival across various age group in test set
Survival
No
Yes
Count
Age
0
10
20
30
40
50
60
70

End of Code

DAL assignment Naive Bayes (revised)

Importing Libraries

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
!pip install kmodes
from kmodes.kprototypes import KPrototypes
from sklearn.naive_bayes import CategoricalNB,ComplementNB,GaussianNB,BernoulliNB
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from sklearn.model_selection import train_test_split,StratifiedKFold,cross_val_scor
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix,roc_curve, auc
import copy
from sklearn.ensemble import VotingClassifier
from sklearn.pipeline import Pipeline
```

```
Collecting kmodes
  Downloading kmodes-0.12.2-py2.py3-none-any.whl (20 kB)
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.10/dist-pac
kages (from kmodes) (1.23.5)
Requirement already satisfied: scikit-learn>=0.22.0 in /usr/local/lib/python3.10/d
ist-packages (from kmodes) (1.2.2)
Requirement already satisfied: scipy>=0.13.3 in /usr/local/lib/python3.10/dist-pac
kages (from kmodes) (1.11.3)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-pack
ages (from kmodes) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/d
ist-packages (from scikit-learn>=0.22.0->kmodes) (3.2.0)
Installing collected packages: kmodes
Successfully installed kmodes-0.12.2
```

Load the data

Note: This Data does not have the column names. So we will set the column names as per the reference PDF

In [ ]:
```
Cols=['Age', 'Workclass', 'fnlwgt', 'Education', 'Education num',
       'Marital status', 'Occupation', 'Relationship', 'Race', 'Sex',
       'Capital gain', 'Capital loss', 'Hours per week', 'Native country',
       'Income']
Data=pd.read_excel('/content/drive/MyDrive/DAL dataset/Assignment 3/adult.xlsx',hea
Data = Data.replace(r"^ +| +$", r"", regex=True)#for eliminating leading or trainir
```

Studying the data

In [ ]:
```
Data.head()
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40

In [ ]:
```
Data.columns=Cols
```

In [ ]:
```
Data.head()
```

Out[ ]:

	Age	Workclass	fnlwgt	Education	Education num	Marital status	Occupation	Relationship	Race	S
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	M
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	M
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	M
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	M
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Fem

In [ ]:
```
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Age             32561 non-null  int64 
 1   Workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   Education       32561 non-null  object
 4   Education num   32561 non-null  int64 
 5   Marital status  32561 non-null  object
 6   Occupation      32561 non-null  object
 7   Relationship    32561 non-null  object
 8   Race            32561 non-null  object
 9   Sex             32561 non-null  object
 10  Capital gain    32561 non-null  int64 
 11  Capital loss    32561 non-null  int64 
 12  Hours per week  32561 non-null  int64 
 13  Native country  32561 non-null  object
 14  Income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

Apparently no missing values. But there are missing values in the form of ? under object type data

In [ ]:
```
#Categorical columns
Cat_col=['Workclass', 'Education', 'Marital status', 'Occupation', 'Relationship',
```

checking the unique values in the categorical columns

In [ ]:
```
for cat in Cat_col:
  print(Data[cat].value_counts())
```

```
Private             22696
Self-emp-not-inc     2541
Local-gov            2093
?                    1836
State-gov            1298
Self-emp-inc         1116
Federal-gov           960
Without-pay            14
Never-worked            7
Name: Workclass, dtype: int64
HS-grad         10501
Some-college     7291
Bachelors        5355
Masters          1723
Assoc-voc        1382
11th             1175
Assoc-acdm       1067
10th              933
7th-8th           646
Prof-school       576
9th               514
12th              433
Doctorate         413
5th-6th           333
1st-4th           168
Preschool          51
Name: Education, dtype: int64
Married-civ-spouse       14976
Never-married            10683
Divorced                  4443
Separated                 1025
Widowed                    993
Married-spouse-absent      418
Married-AF-spouse           23
Name: Marital status, dtype: int64
Prof-specialty       4140
Craft-repair         4099
Exec-managerial      4066
Adm-clerical         3770
Sales                3650
Other-service        3295
Machine-op-inspct    2002
?                    1843
Transport-moving     1597
Handlers-cleaners    1370
Farming-fishing       994
Tech-support          928
Protective-serv       649
Priv-house-serv       149
Armed-Forces            9
Name: Occupation, dtype: int64
Husband           13193
Not-in-family      8305
Own-child          5068
Unmarried          3446
Wife               1568
Other-relative      981
Name: Relationship, dtype: int64
White                27816
Black                 3124
Asian-Pac-Islander    1039
Amer-Indian-Eskimo     311
Other                  271
Name: Race, dtype: int64
```

```
Male      21790
Female    10771
Name: Sex, dtype: int64
United-States                 29170
Mexico                          643
?                               583
Philippines                     198
Germany                         137
Canada                          121
Puerto-Rico                     114
El-Salvador                     106
India                           100
Cuba                             95
England                          90
Jamaica                          81
South                            80
China                            75
Italy                            73
Dominican-Republic               70
Vietnam                          67
Guatemala                        64
Japan                            62
Poland                           60
Columbia                         59
Taiwan                           51
Haiti                            44
Iran                             43
Portugal                         37
Nicaragua                        34
Peru                             31
France                           29
Greece                           29
Ecuador                          28
Ireland                          24
Hong                             20
Cambodia                         19
Trinadad&Tobago                  19
Laos                             18
Thailand                         18
Yugoslavia                       16
Outlying-US(Guam-USVI-etc)       14
Honduras                         13
Hungary                          13
Scotland                         12
Holand-Netherlands                1
Name: Native country, dtype: int64
<=50K    24720
>50K      7841
Name: Income, dtype: int64
```

Only these 3 columns have ? marks 'Workclass', 'Occupations', 'Native country'

Modifying the Income column entries for plotting purpose

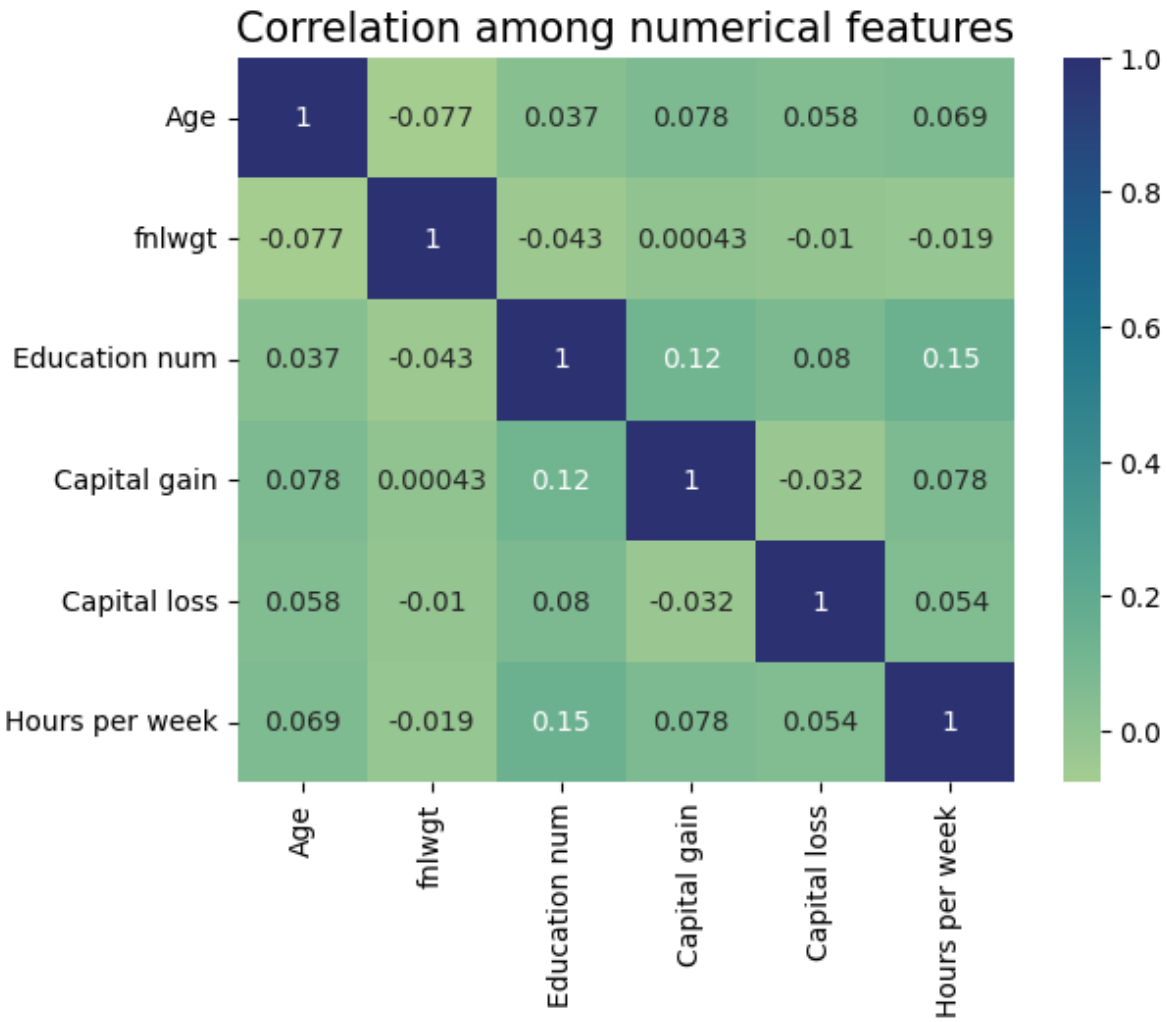
In [ ]:
```
for i in range(len(Data['Income'])):
  Data.loc[i,'Income']=Data.loc[i,'Income']+' USD'
```

In [ ]:
```
#We will replace the '?' marks using NAN
cols_with_qnmark=['Workclass','Occupation','Native country']
for col in cols_with_qnmark:
  Data[col].replace('?',np.NaN,inplace=True)
Data.info()#We are replacing this with Nan so that it does not interfere with plott
```

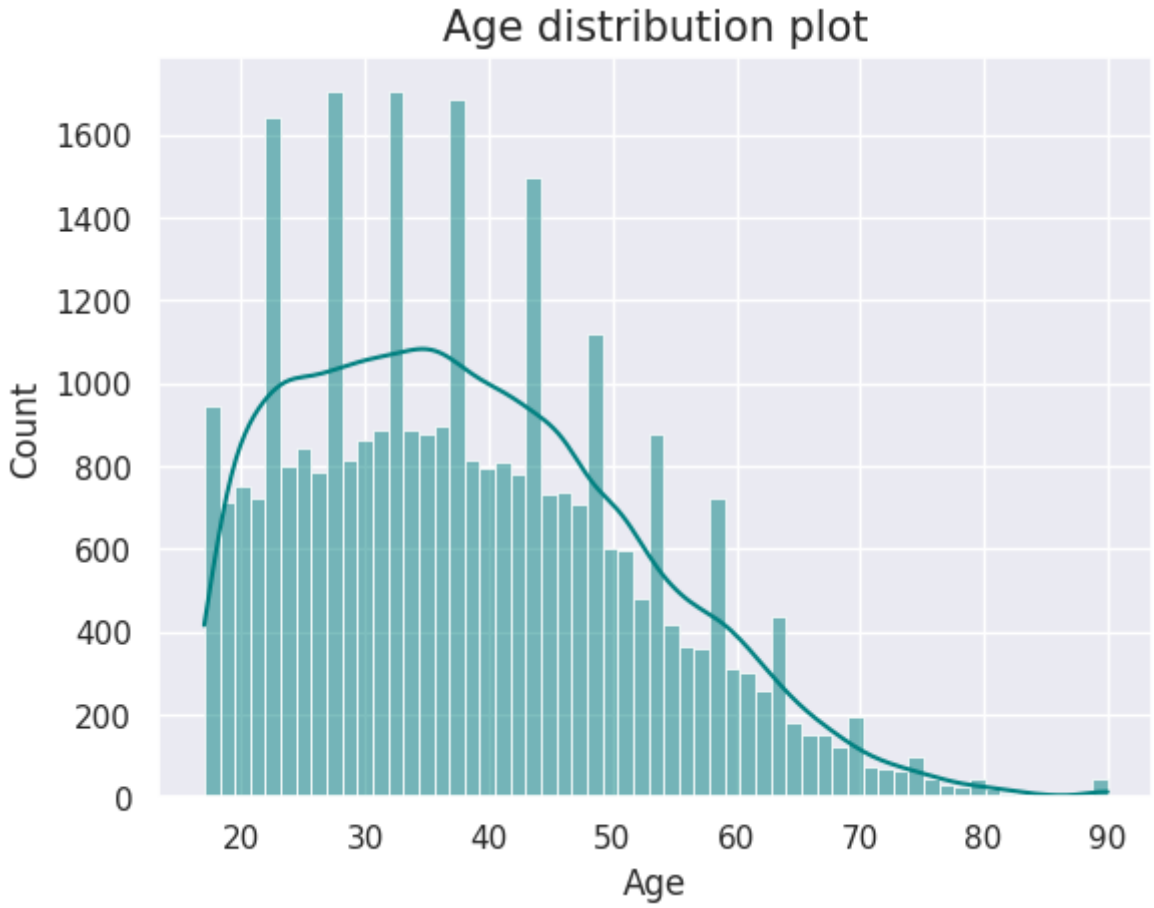
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Age             32561 non-null  int64 
 1   Workclass       30725 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   Education       32561 non-null  object
 4   Education num   32561 non-null  int64 
 5   Marital status  32561 non-null  object
 6   Occupation      30718 non-null  object
 7   Relationship    32561 non-null  object
 8   Race            32561 non-null  object
 9   Sex             32561 non-null  object
 10  Capital gain    32561 non-null  int64 
 11  Capital loss    32561 non-null  int64 
 12  Hours per week  32561 non-null  int64 
 13  Native country  31978 non-null  object
 14  Income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

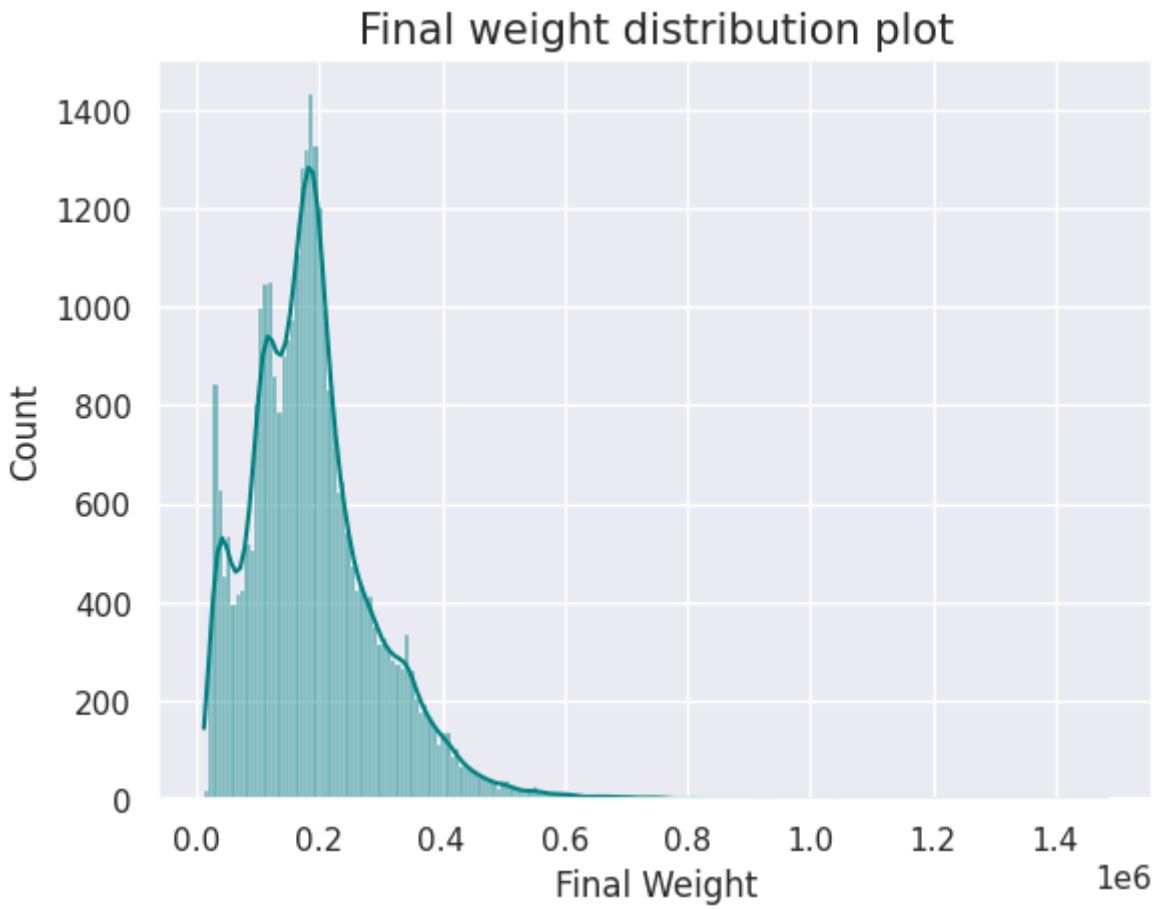
These revels how many missing entries are there in the data ,only in those three categorical columns namely workclass,occupation, native country

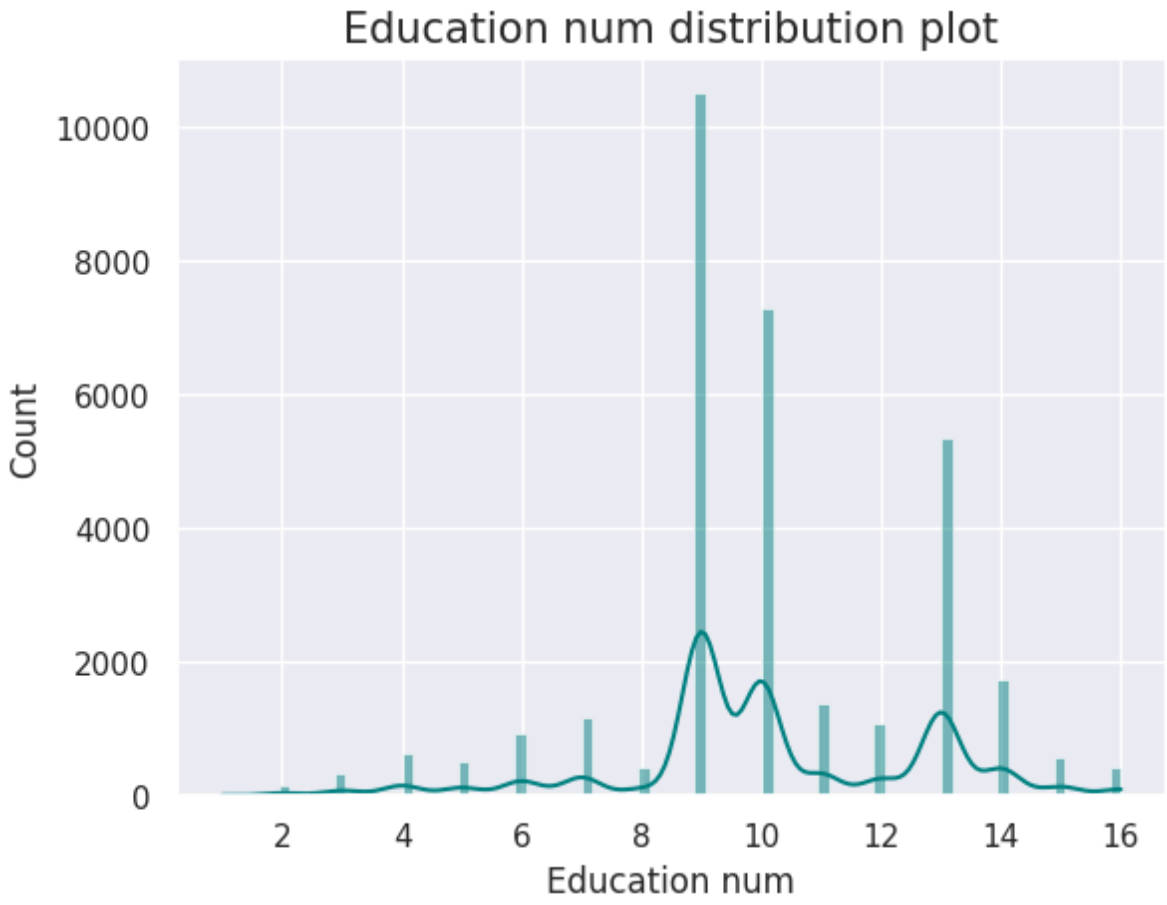
In [ ]:
```
#Check correlation
Correlation=Data.corr(numeric_only=True)
sns.heatmap(Correlation,annot=True,cmap='crest')
plt.title("Correlation among numerical features",fontsize=15)
plt.show()
```

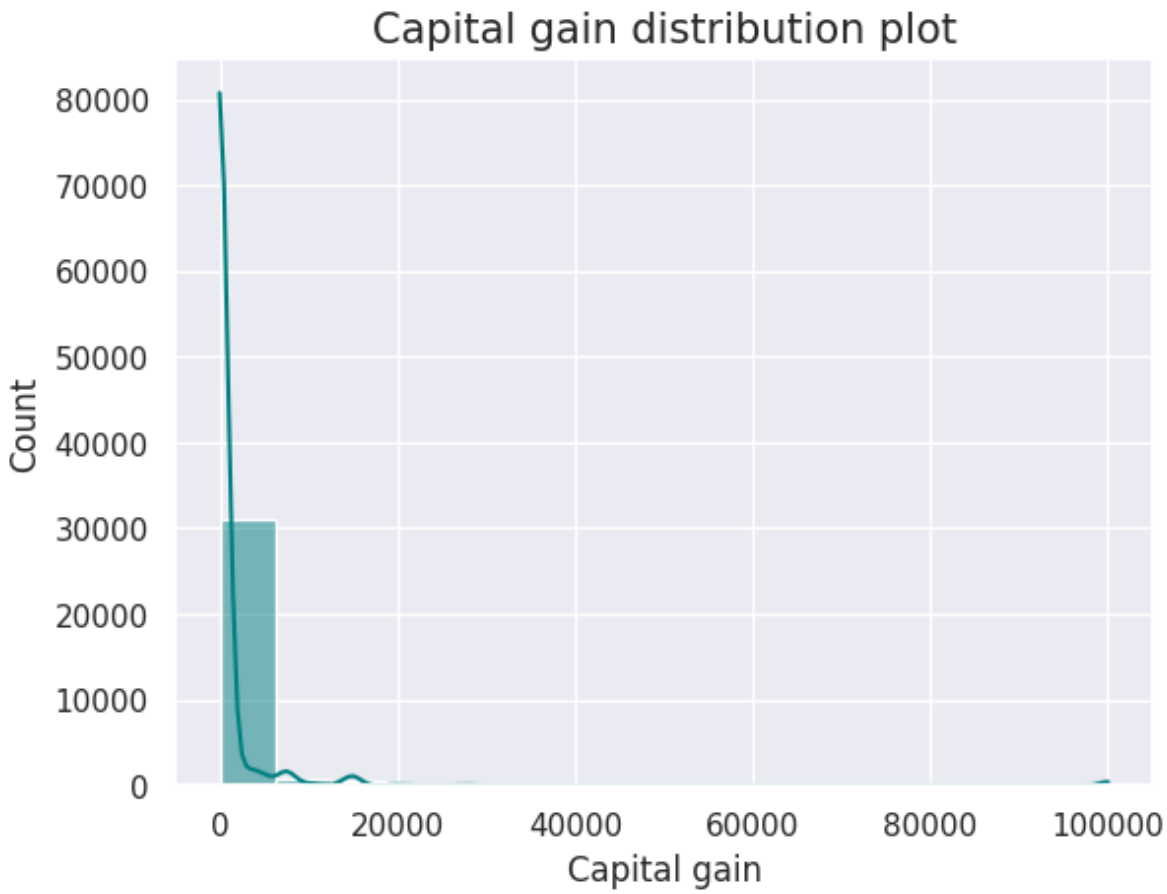


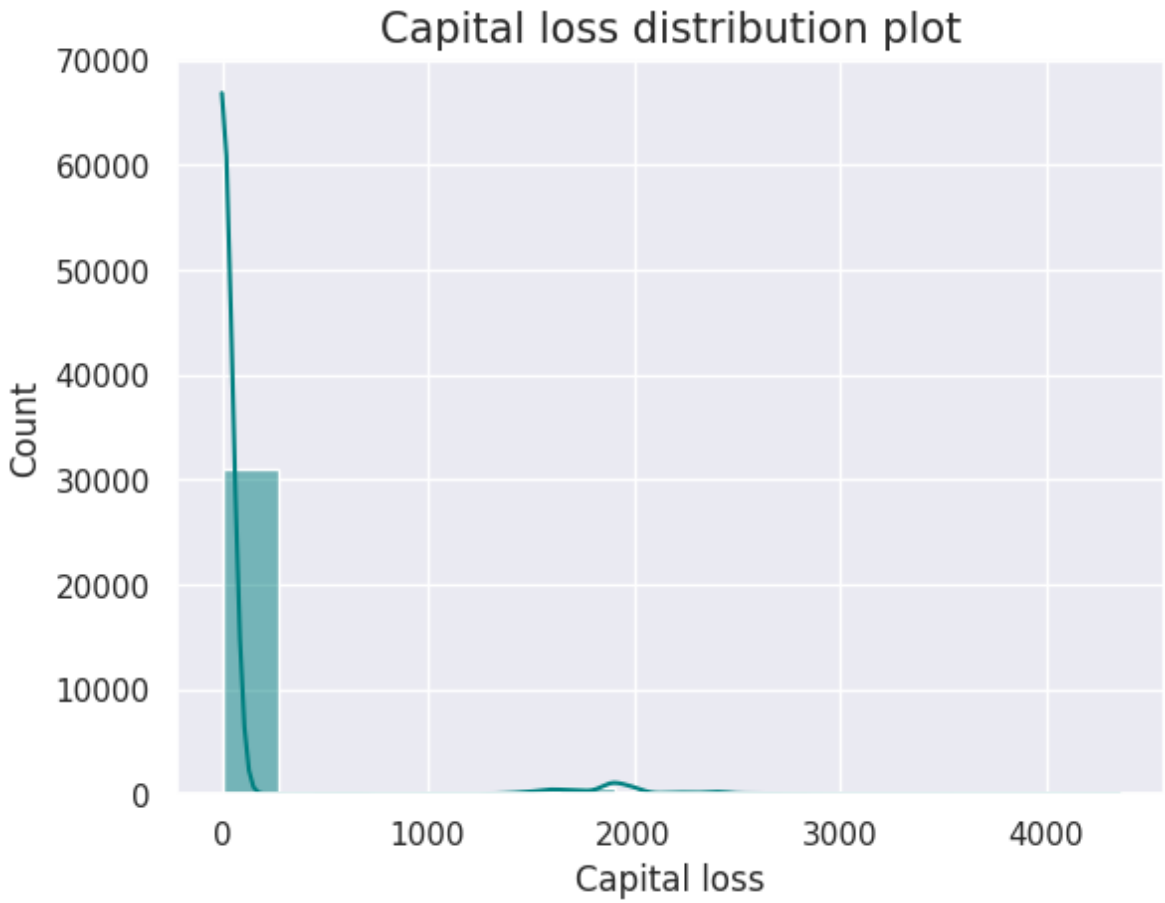
In [ ]:
```
#Check the distributions
Num_cols=['Age','fnlwgt','Education num','Capital gain','Capital loss','Hours per w
sns.set()
for num in Num_cols:
  if num=='fnlwgt':
    plt.xlabel('Final Weight')
    plt.title("Final weight distribution plot",fontsize=15)
  elif num=='Education_num':
    plt.xlabel('Number of years of education')
    plt.title("Education duration distribution plot",fontsize=15)
  else:
    plt.title(f"{num} distribution plot",fontsize=15)
  sns.histplot(data=Data,x=num,kde=True,color='teal')
  plt.show()
```

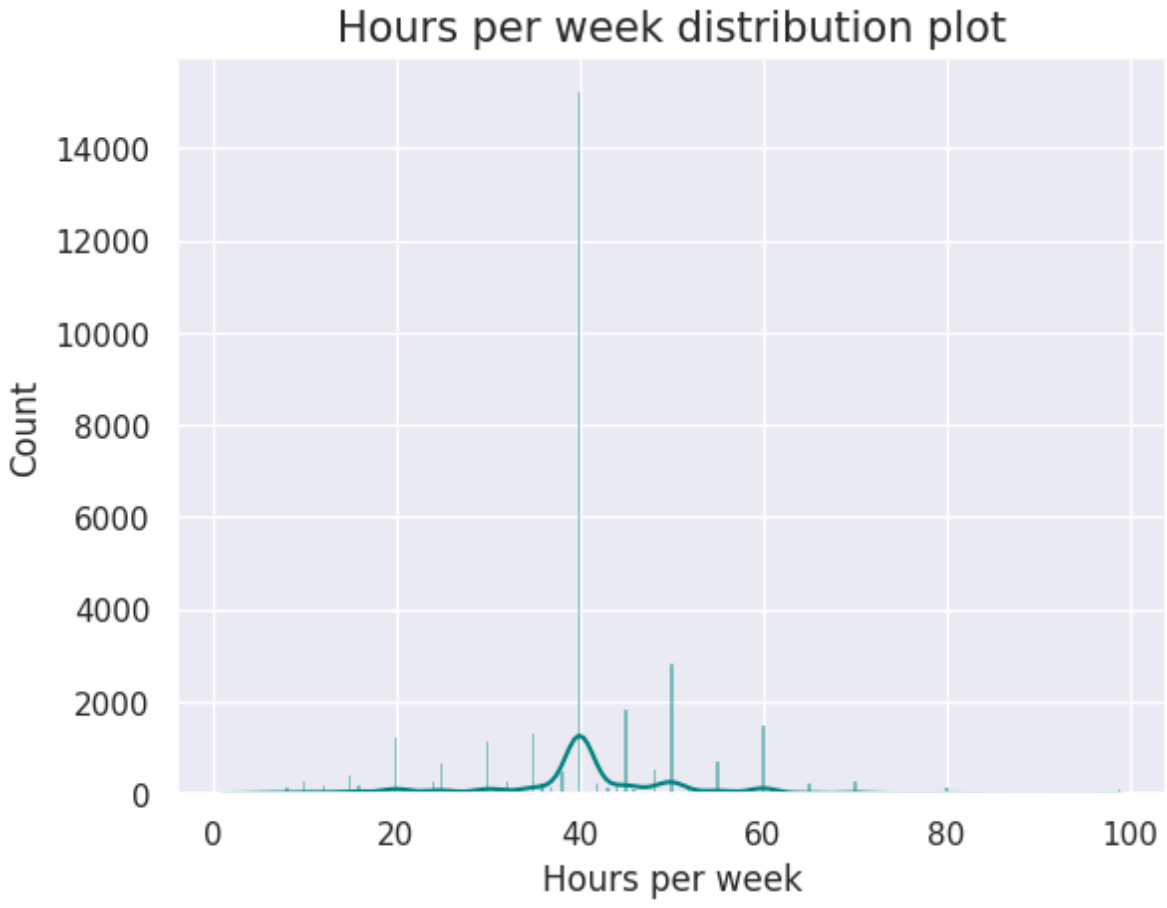

Age distribution plot
Count
0
200
400
600
800
1000
1200
1400
1600
20
30
40
50
60
70
80
90
Age


Final weight distribution plot
Count
0
200
400
600
800
1000
1200
1400
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e6
Final Weight


Education num distribution plot
Count
10000
8000
6000
4000
2000
0
2
4
6
8
10
12
14
16
Education num

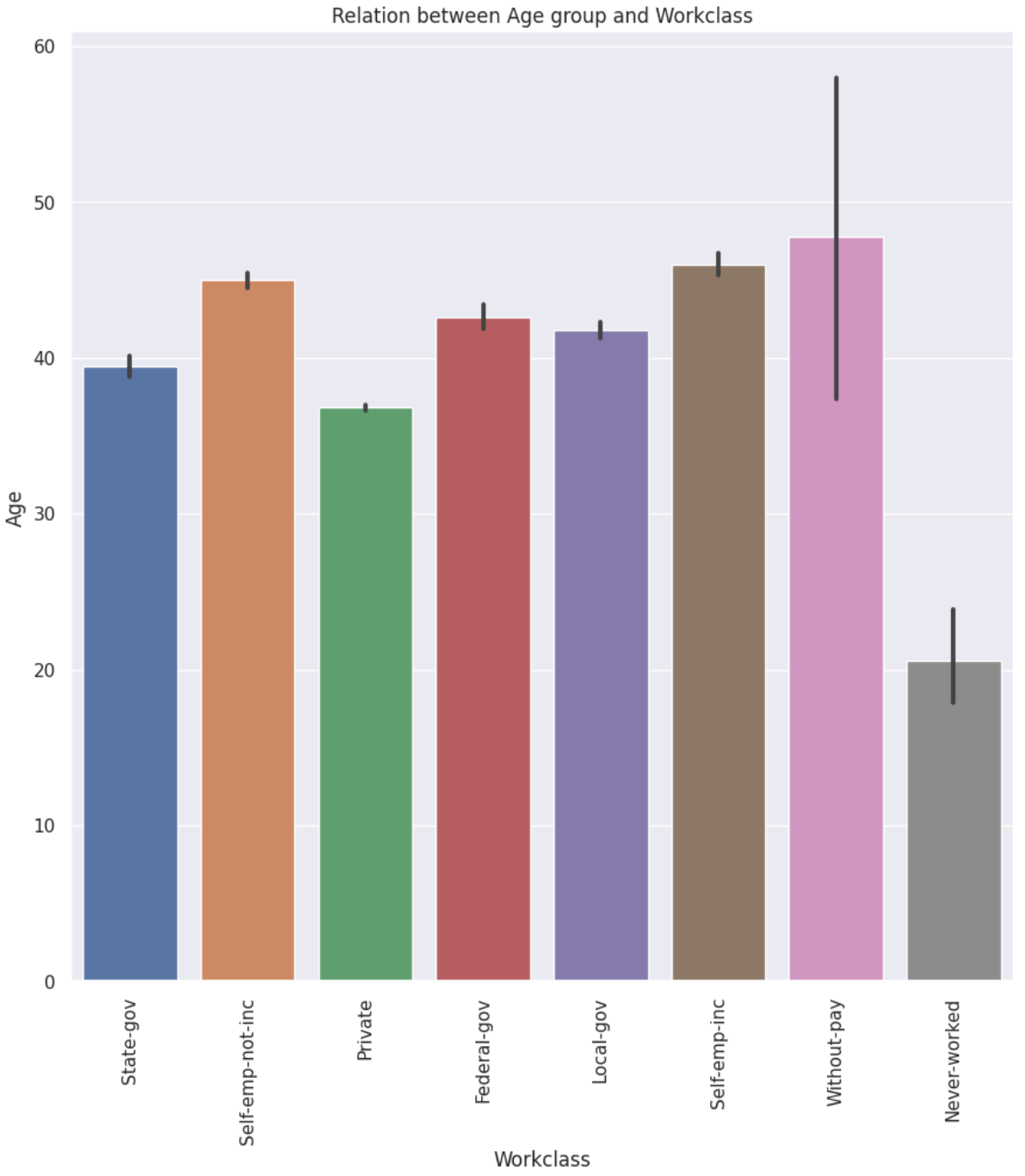

Capital gain distribution plot
Count
80000
70000
60000
50000
40000
30000
20000
10000
0
0
20000
40000
60000
80000
100000
Capital gain

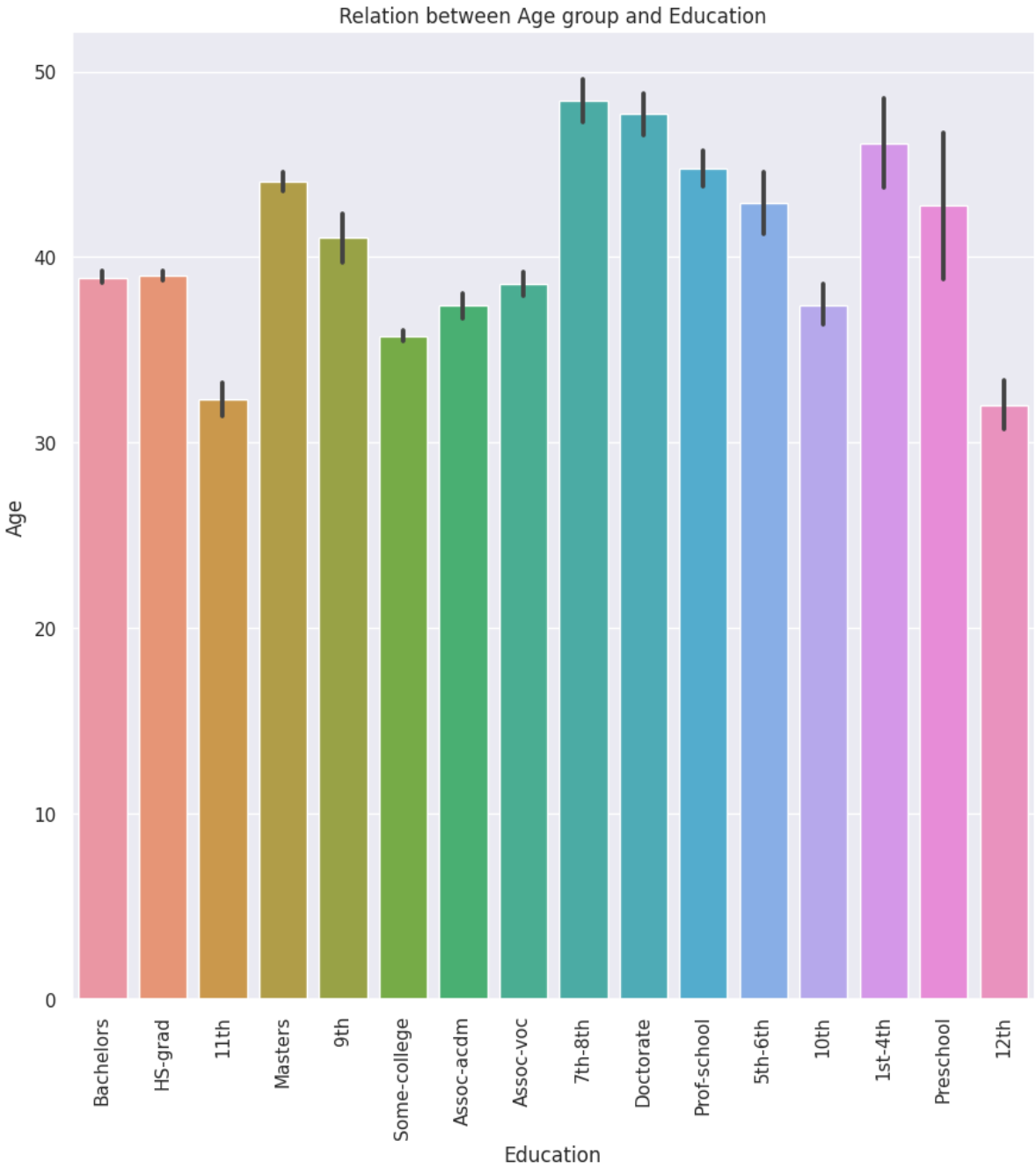


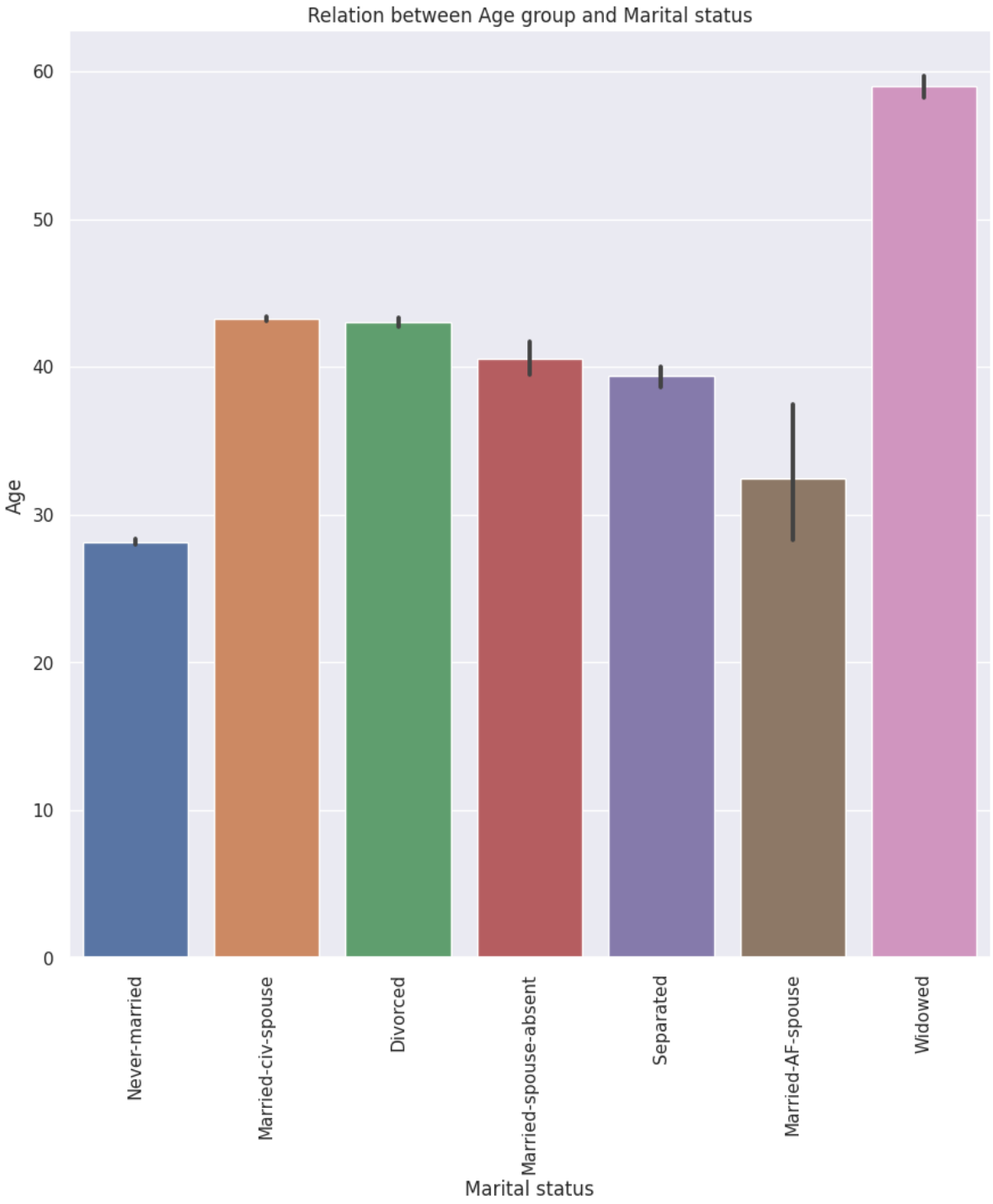


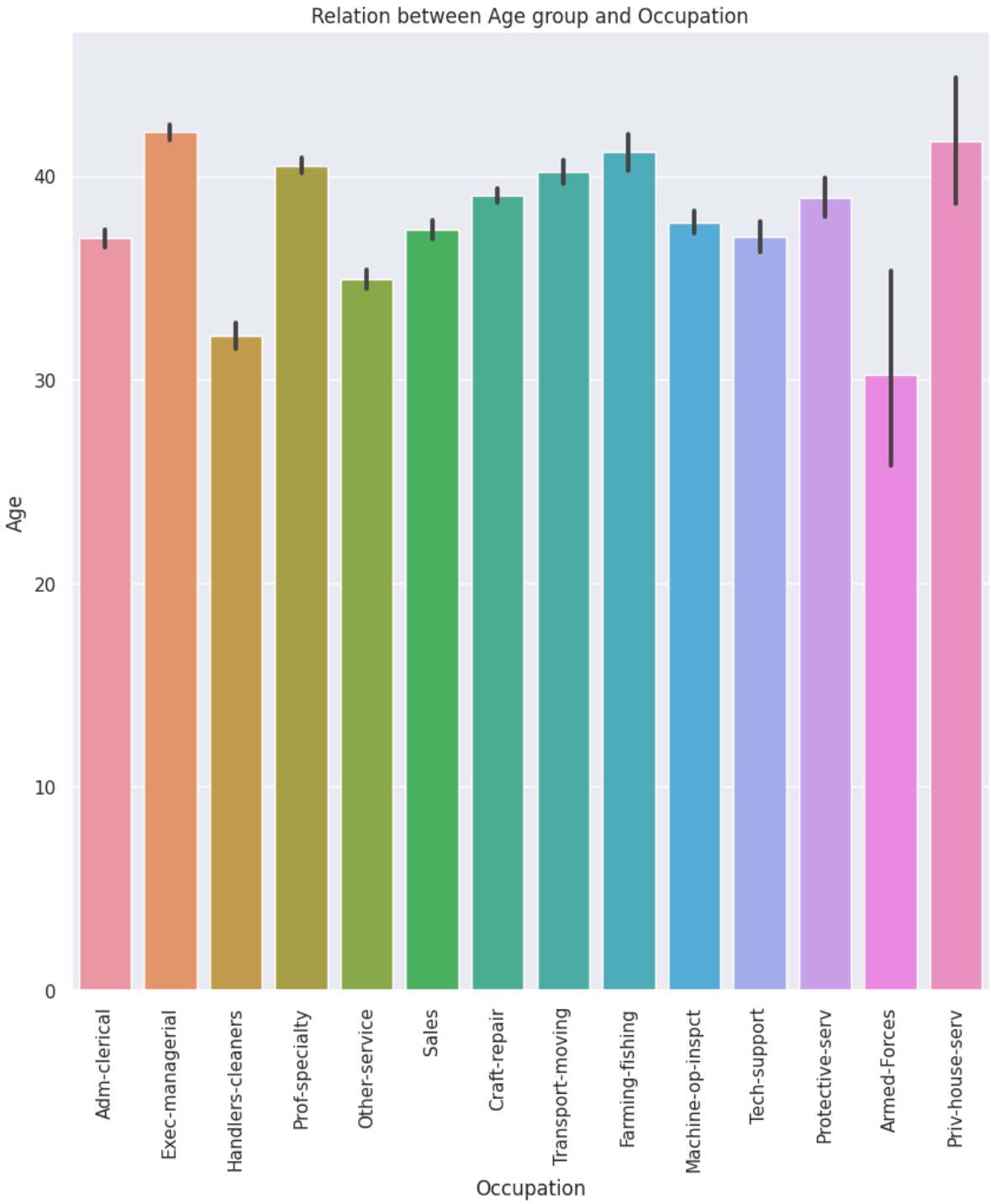
In [ ]:
```
sns.set()
#check the relation between age and categorical columns
for cat_col in Cat_col:
  sns.catplot(data=Data,x=cat_col,y='Age',kind='bar',height=9)
  plt.title(f"Relation between Age group and {cat_col}")
```

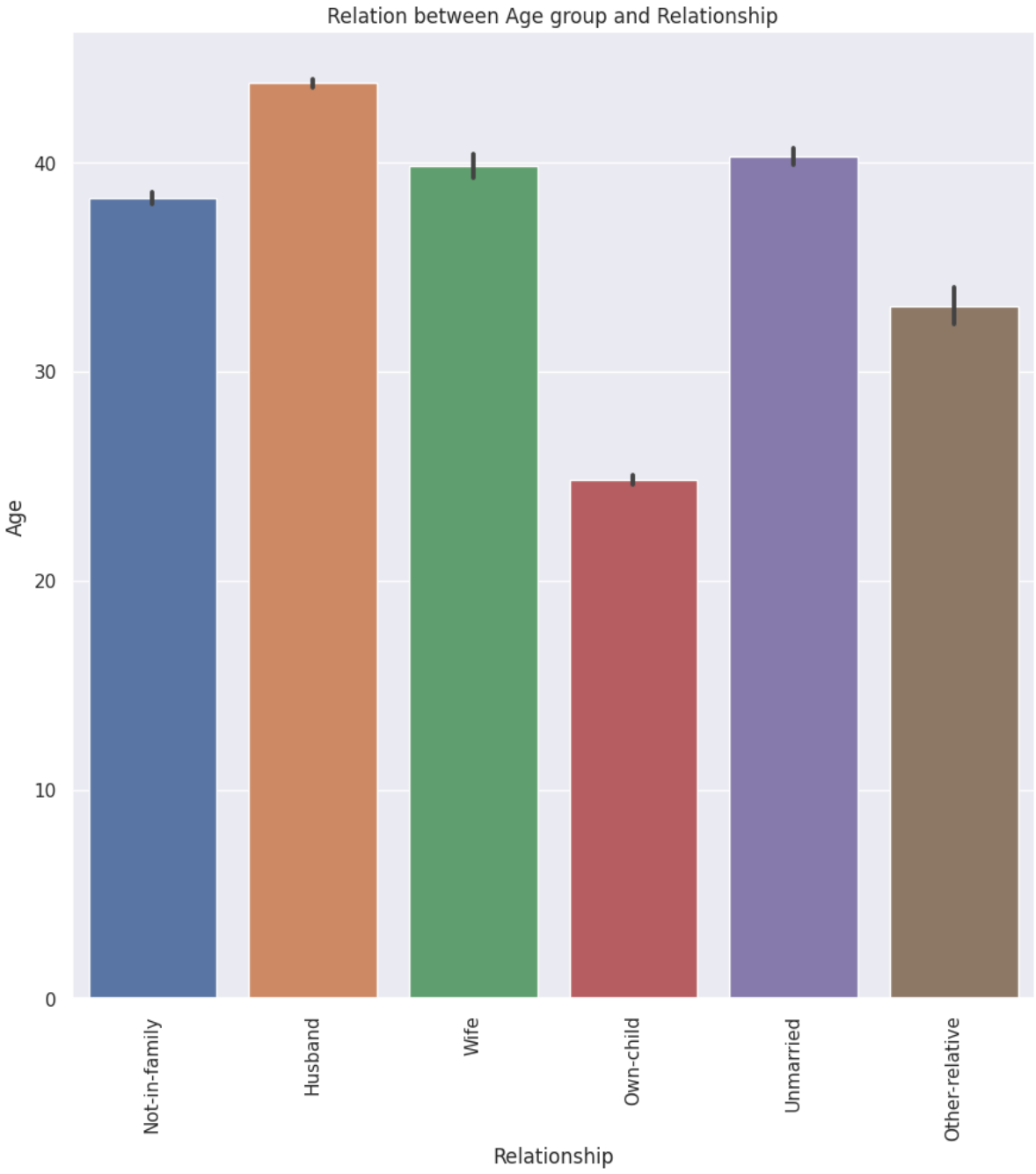
```
plt.xticks(rotation=90)
plt.show()
```

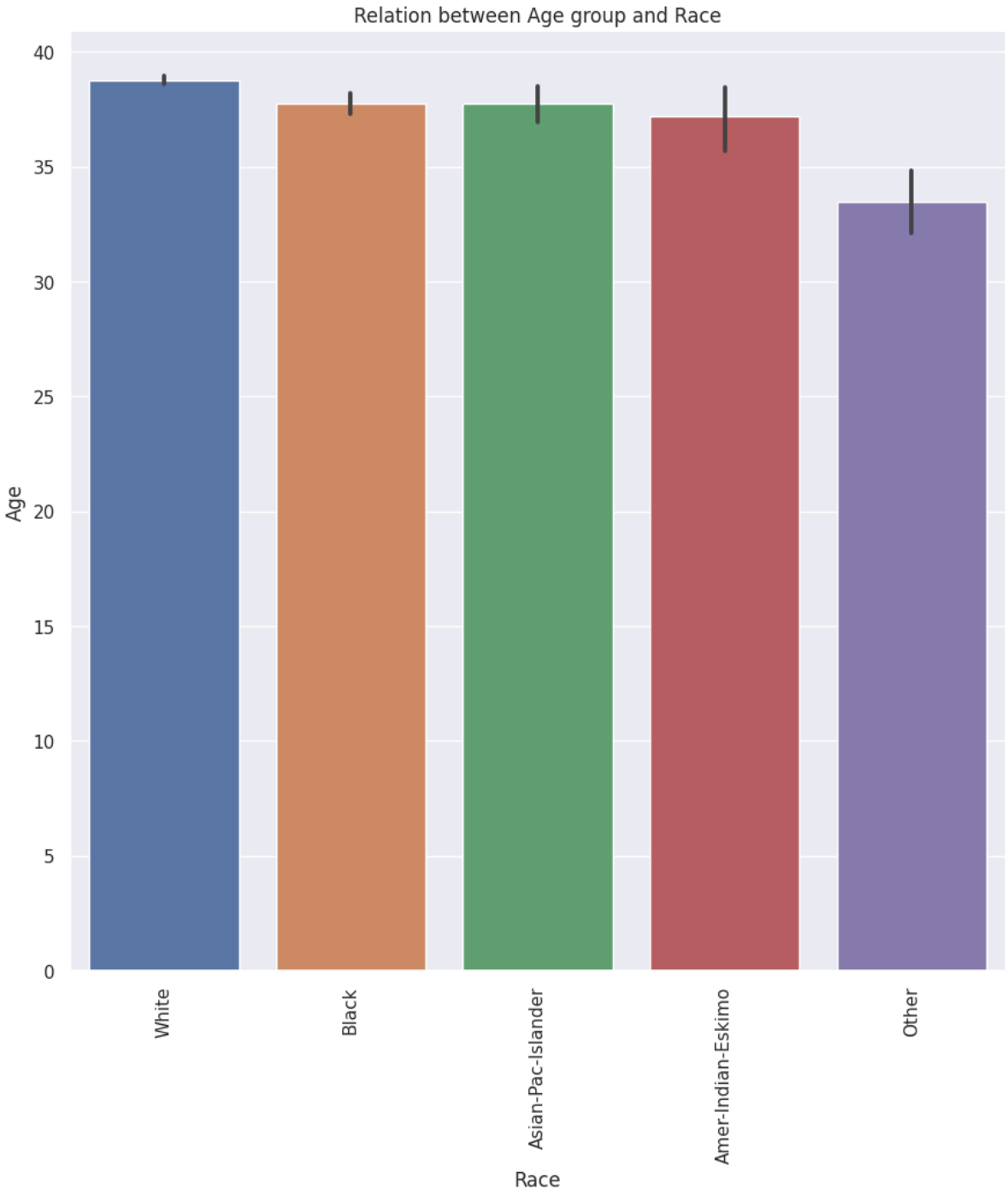


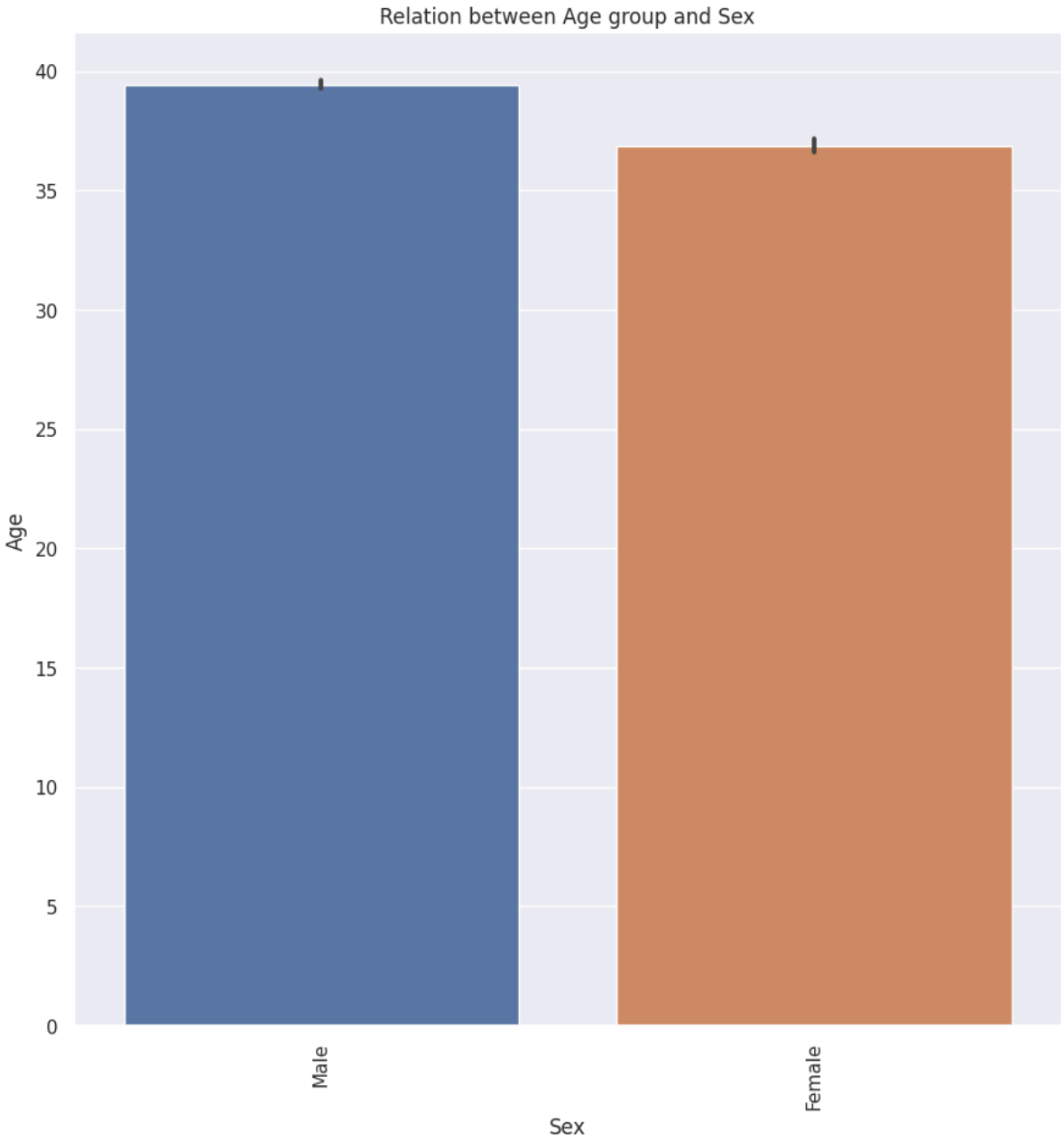

Relation between Age group and Education
Age
0
10
20
30
40
50
Bachelors
HS-grad
11th
Masters
9th
Some-college
Assoc-acdm
Assoc-voc
7th-8th
Doctorate
Prof-school
5th-6th
10th
1st-4th
Preschool
12th
Education

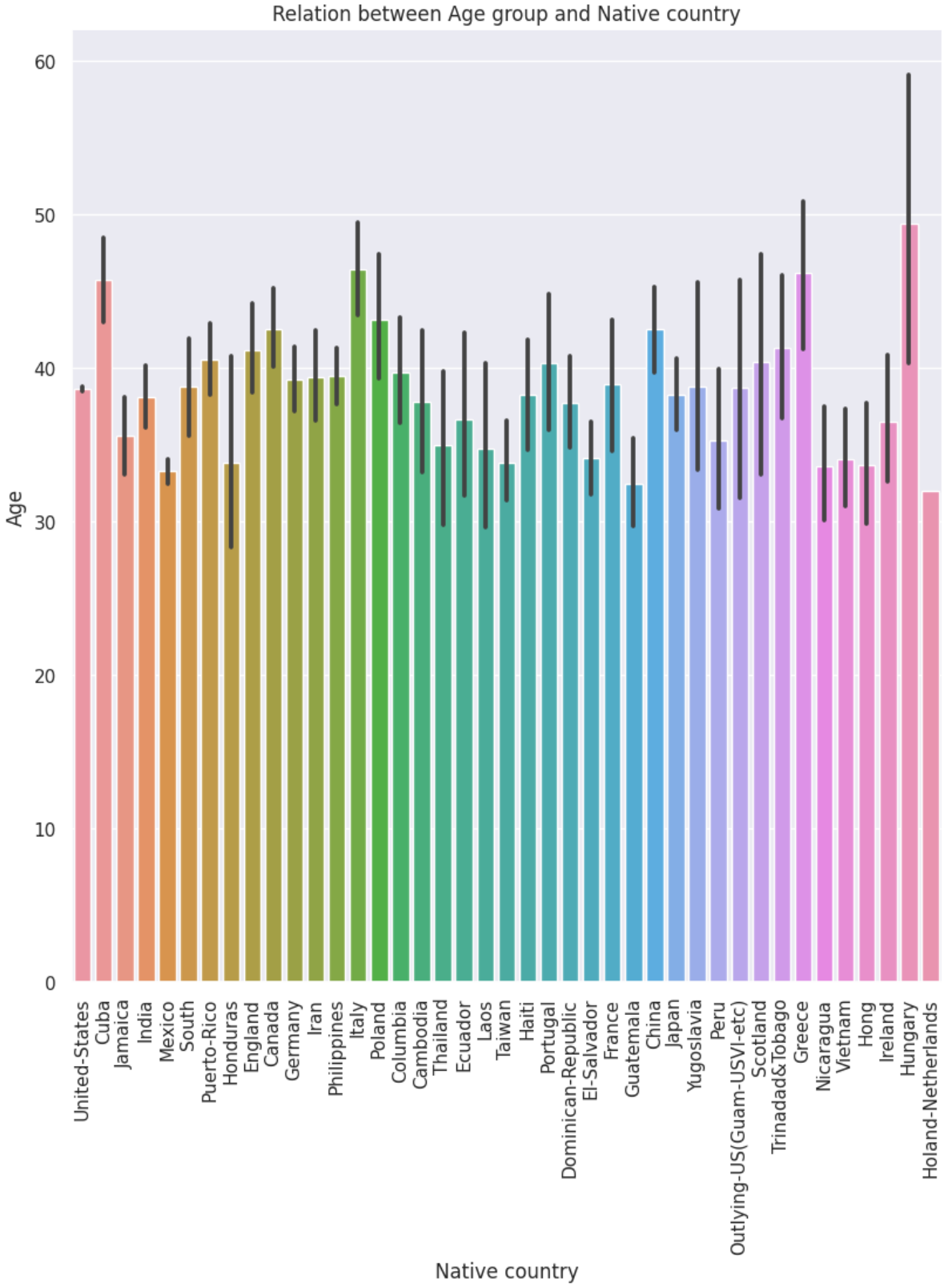

Relation between Age group and Marital status
Age
60
50
40
30
20
10
0
Never-married
Married-civ-spouse
Divorced
Married-spouse-absent
Separated
Married-AF-spouse
Widowed
Marital status

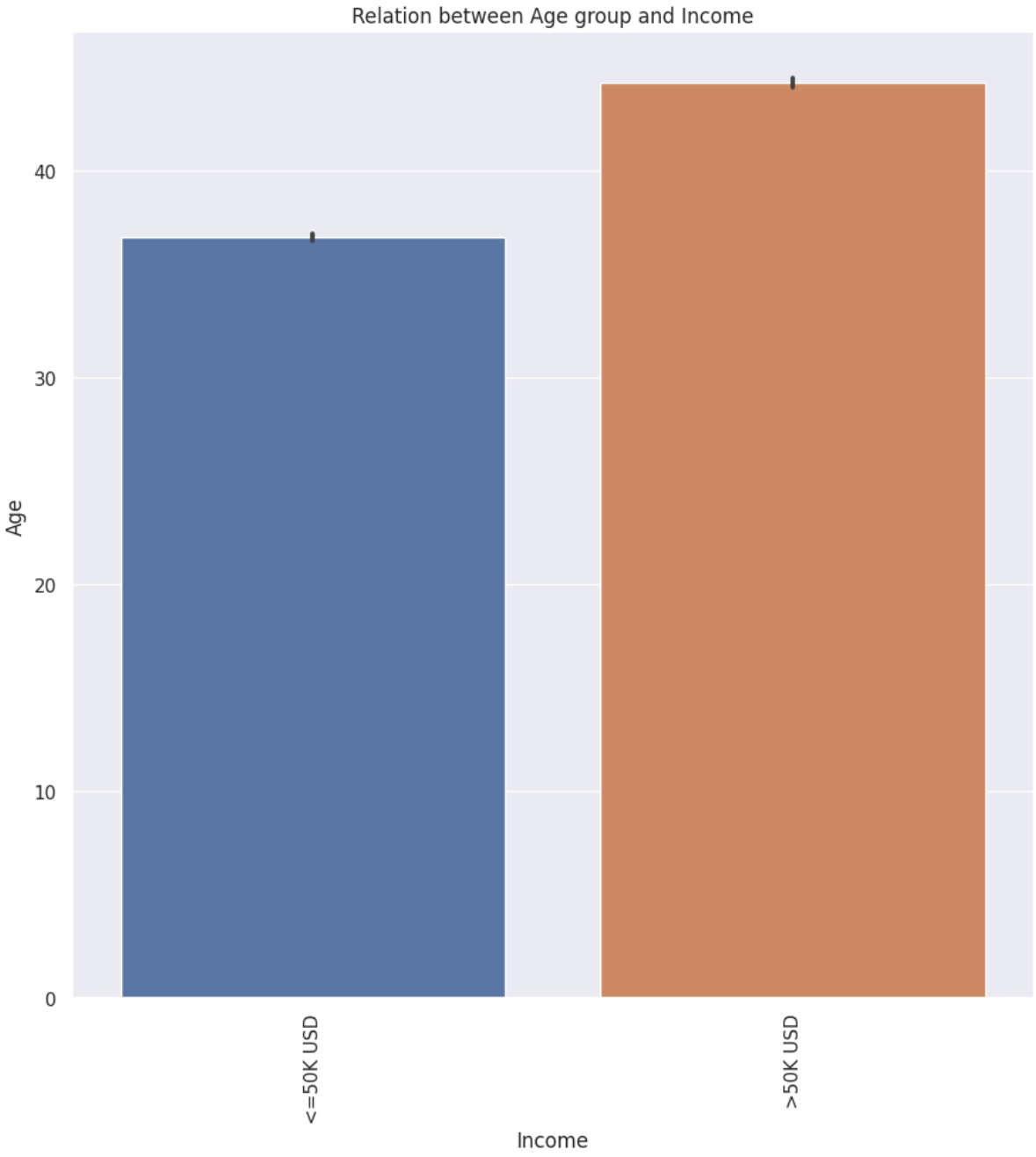

Relation between Age group and Occupation
Age
0
10
20
30
40
Adm-clerical
Exec-managerial
Handlers-cleaners
Prof-specialty
Other-service
Sales
Craft-repair
Transport-moving
Farming-fishing
Machine-op-inspct
Tech-support
Protective-serv
Armed-Forces
Priv-house-serv
Occupation


Relation between Age group and Relationship
Age
0
10
20
30
40
Not-in-family
Husband
Wife
Own-child
Unmarried
Other-relative
Relationship


Relation between Age group and Race
Age
Race
0
5
10
15
20
25
30
35
40
White
Black
Asian-Pac-Islander
Amer-Indian-Eskimo
Other

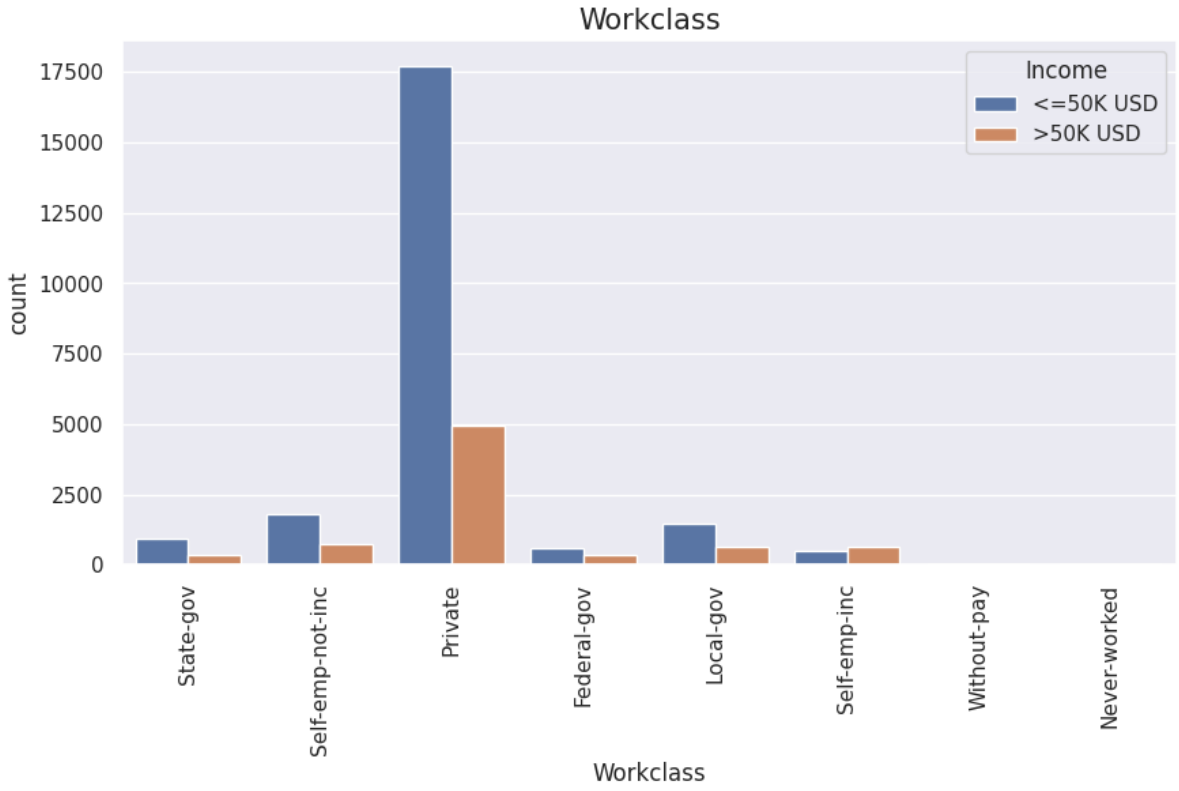

Relation between Age group and Sex
40
35
30
25
20
15
10
5
0
Age
Male
Female
Sex

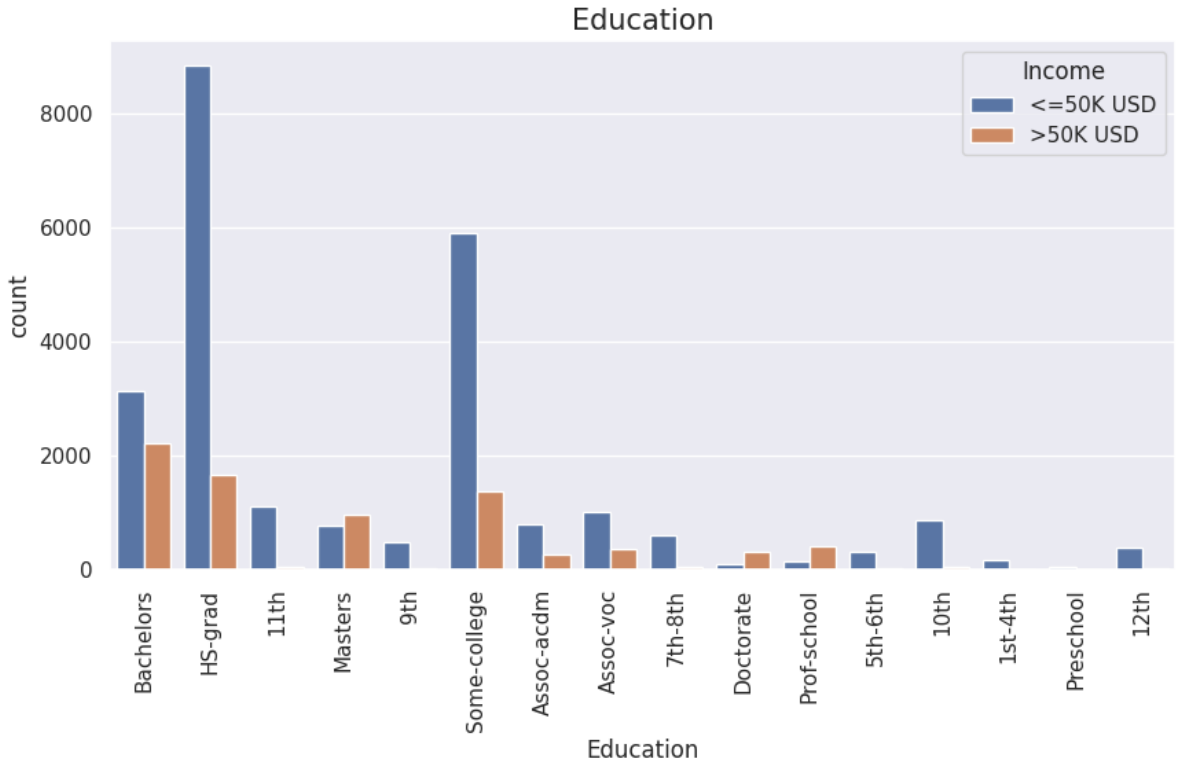

Relation between Age group and Native country
Age
0
10
20
30
40
50
60
United-States
Cuba
Jamaica
India
Mexico
South
Puerto-Rico
Honduras
England
Canada
Germany
Iran
Philippines
Italy
Poland
Columbia
Cambodia
Thailand
Ecuador
Laos
Taiwan
Haiti
Portugal
Dominican-Republic
El-Salvador
France
Guatemala
China
Japan
Yugoslavia
Peru
Outlying-US(Guam-USVI-etc)
Scotland
Trinadad&Tobago
Greece
Nicaragua
Vietnam
Hong
Ireland
Hungary
Holand-Netherlands
Native country

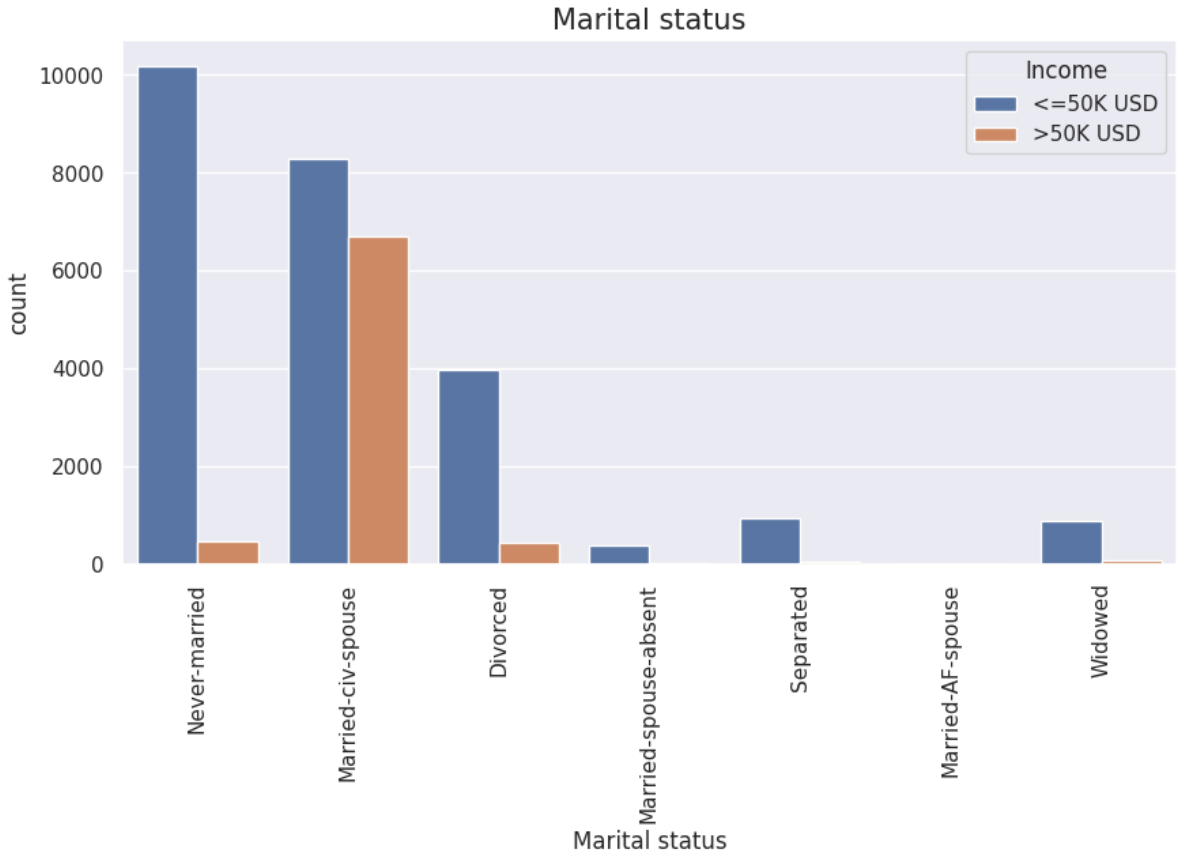


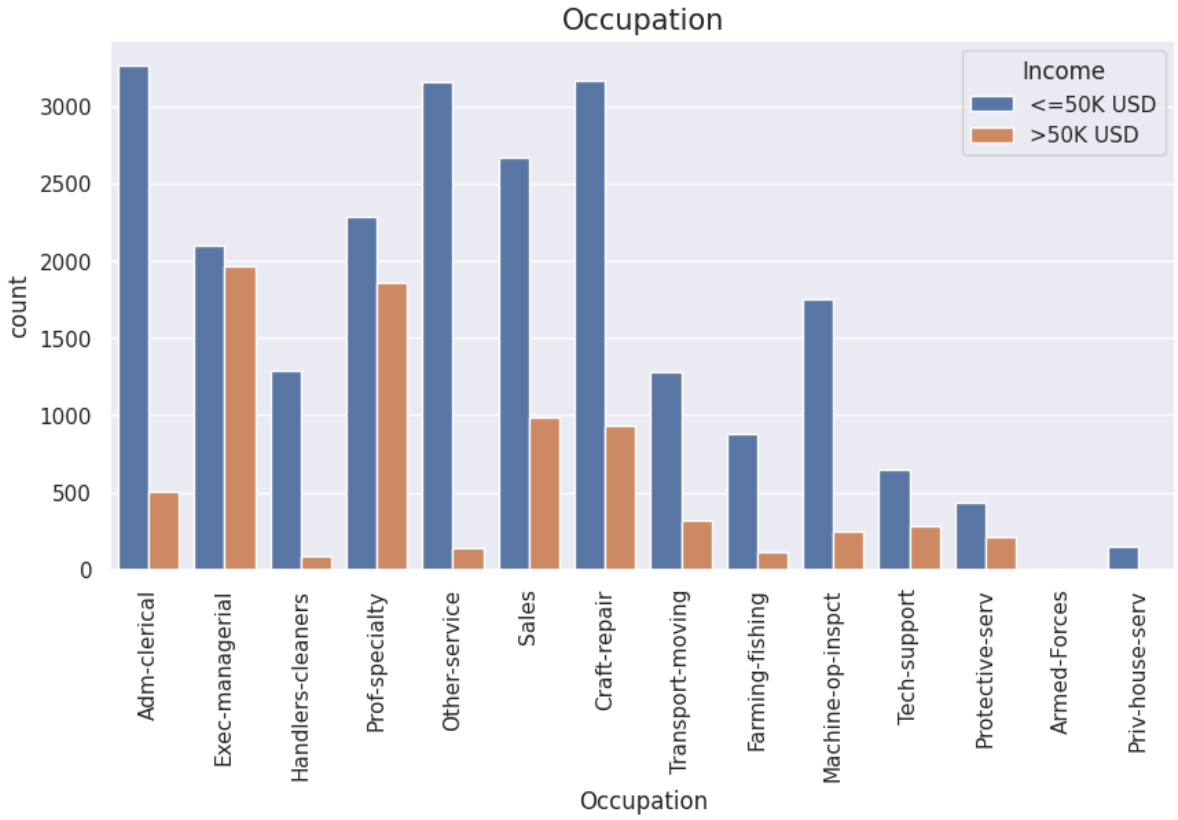
In [ ]:

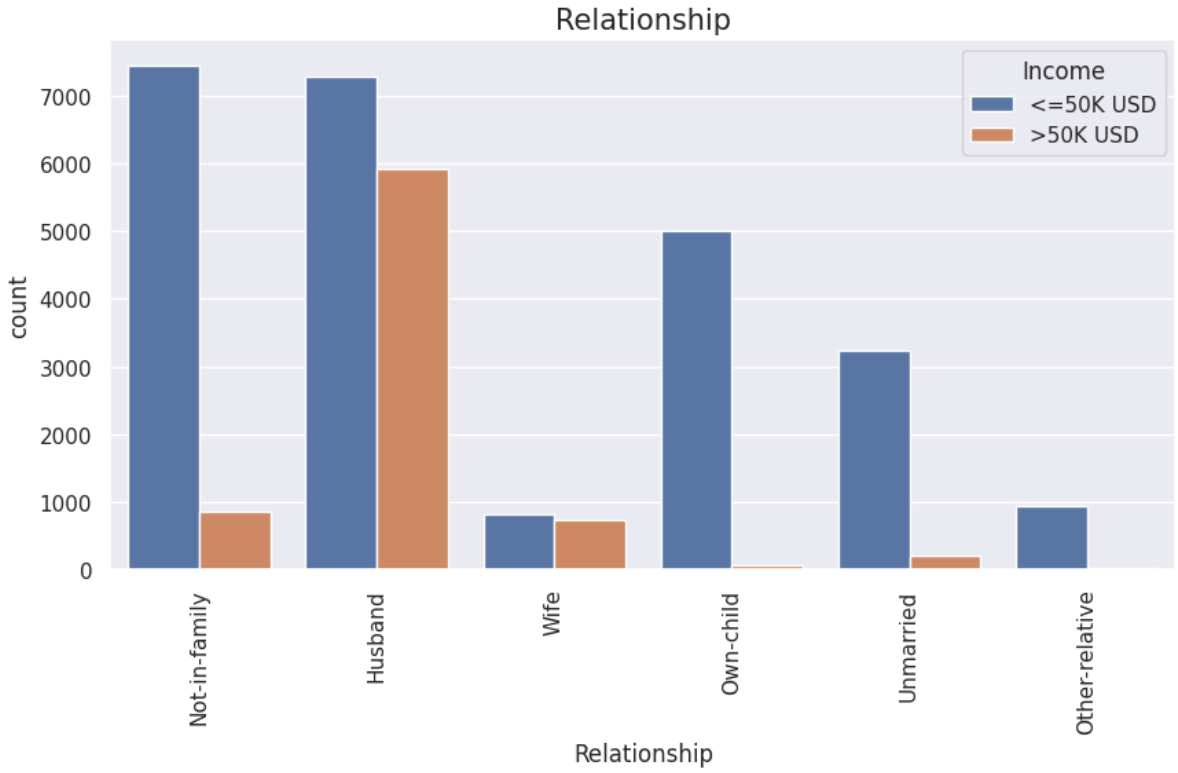
```
sns.set()
#check the bar plots for categorical features
for cat in Cat_col:
  plt.figure(figsize=(10,5))
  sns.countplot(x=cat,data=Data,hue='Income')
  plt.title(cat,fontsize=15)
  plt.xticks(rotation=90)
  plt.show()
```

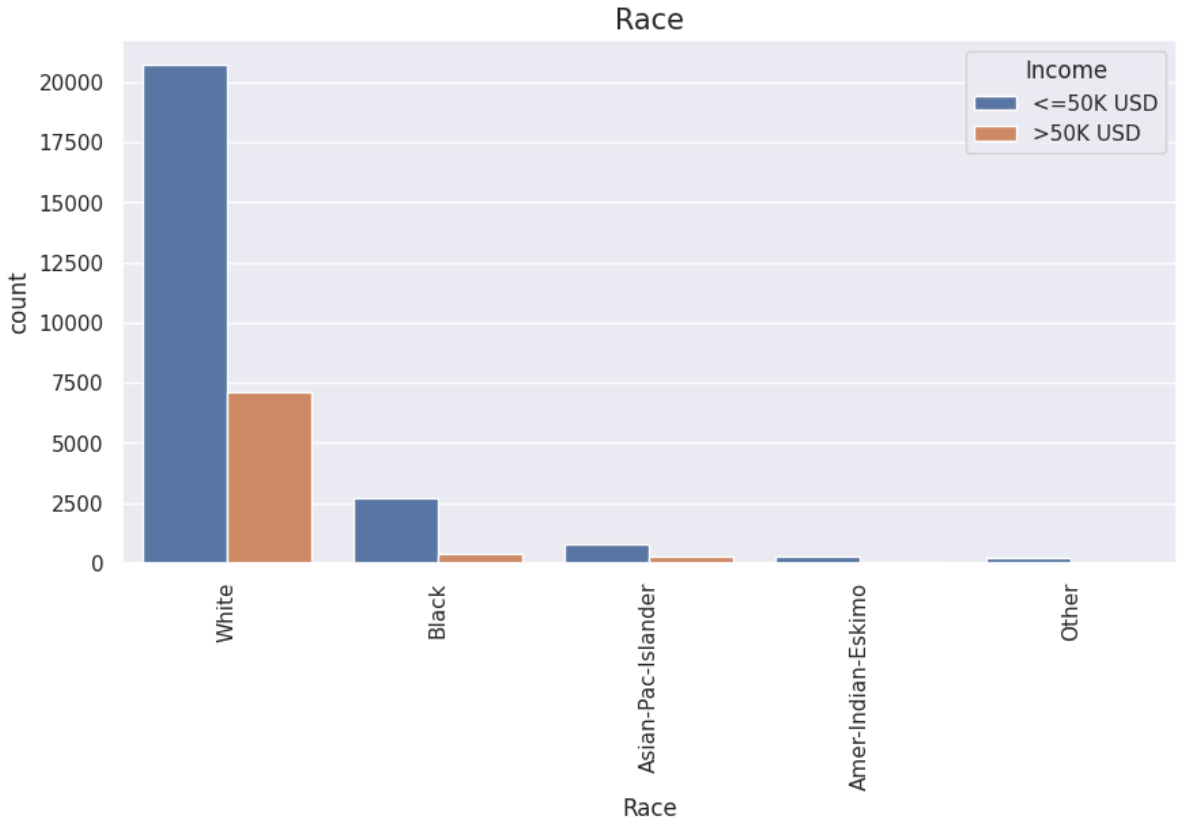

Workclass
Income
<=50K USD
>50K USD
count
17500
15000
12500
10000
7500
5000
2500
0
State-gov
Self-emp-not-inc
Private
Federal-gov
Local-gov
Self-emp-inc
Without-pay
Never-worked
Workclass

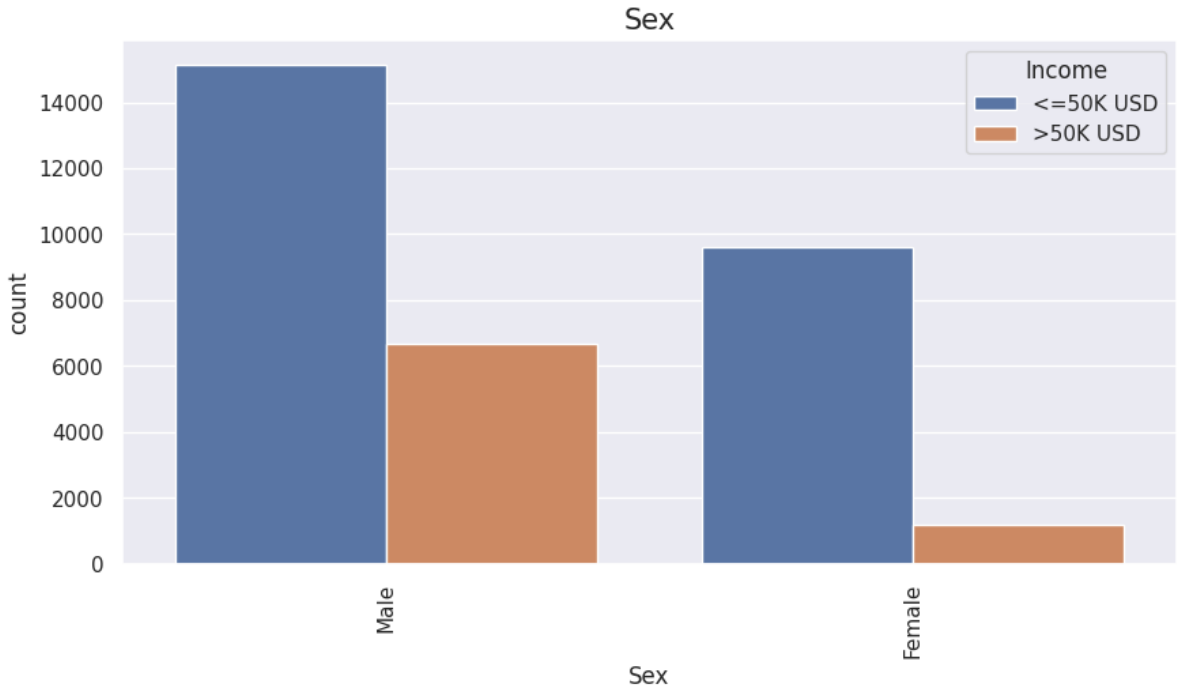

Education
Income
<=50K USD
>50K USD
count
8000
6000
4000
2000
0
Bachelors
HS-grad
11th
Masters
9th
Some-college
Assoc-acdm
Assoc-voc
7th-8th
Doctorate
Prof-school
5th-6th
10th
1st-4th
Preschool
12th
Education

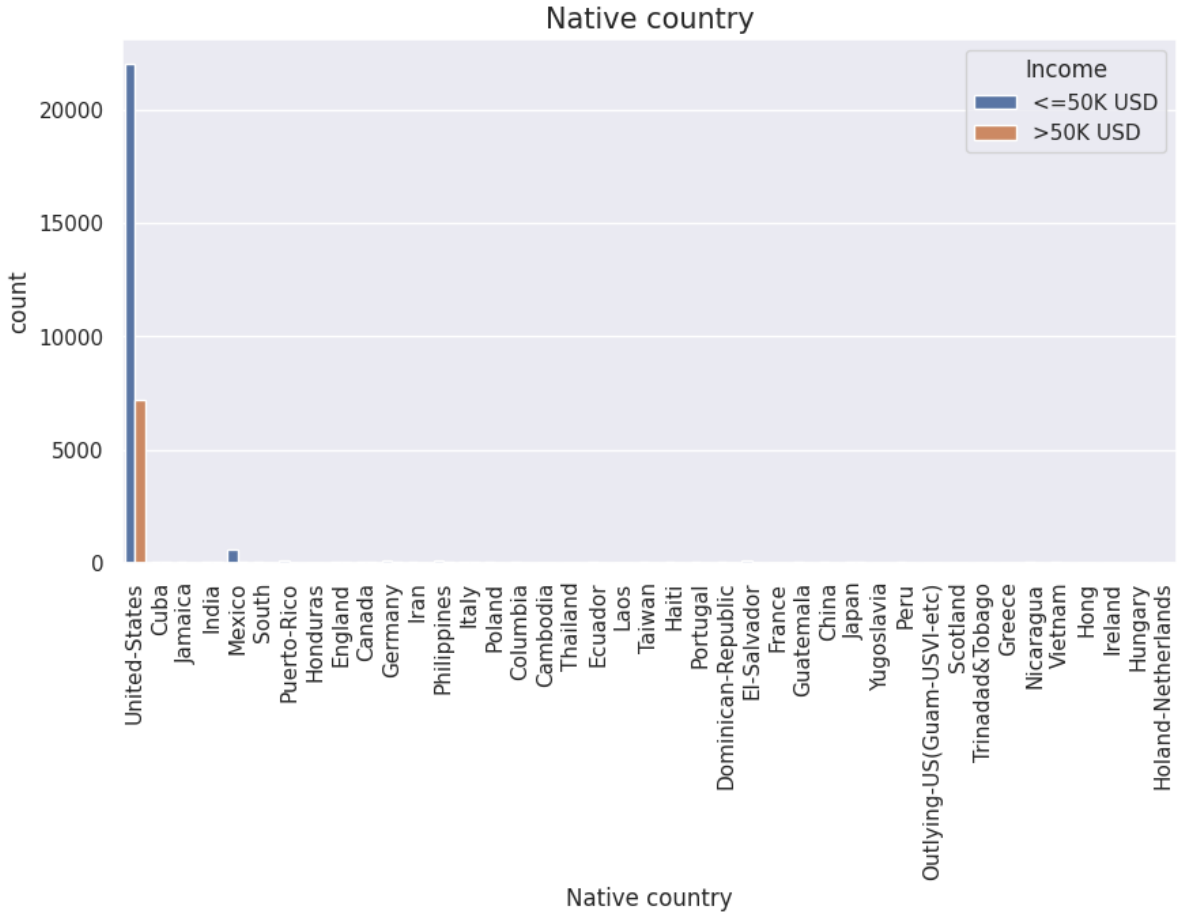

Marital status
Income
<=50K USD
>50K USD
count
10000
8000
6000
4000
2000
0
Never-married
Married-civ-spouse
Divorced
Married-spouse-absent
Separated
Married-AF-spouse
Widowed
Marital status

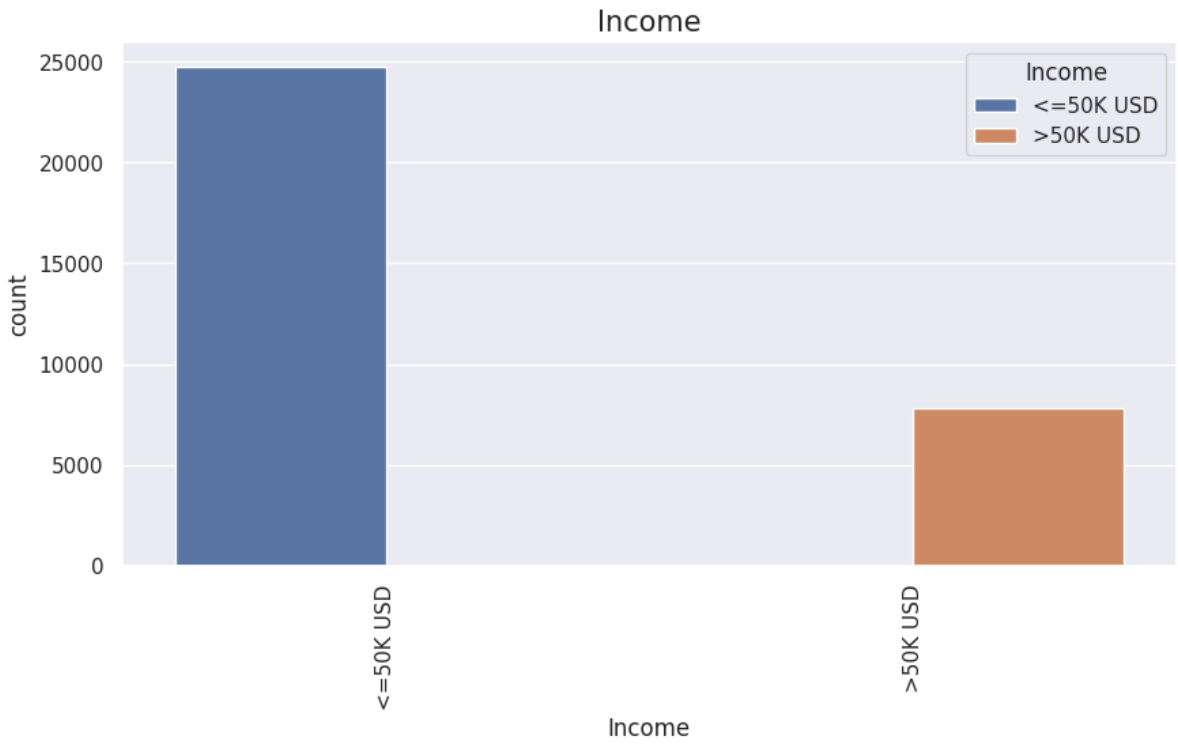

Occupation
Income
<=50K USD
>50K USD
count
3000
2500
2000
1500
1000
500
0
Adm-clerical
Exec-managerial
Handlers-cleaners
Prof-specialty
Other-service
Sales
Craft-repair
Transport-moving
Farming-fishing
Machine-op-inspct
Tech-support
Protective-serv
Armed-Forces
Priv-house-serv
Occupation


Relationship
Income
<=50K USD
>50K USD
count
0
1000
2000
3000
4000
5000
6000
7000
Not-in-family
Husband
Wife
Own-child
Unmarried
Other-relative
Relationship


Race
Income
<=50K USD
>50K USD
count
0
2500
5000
7500
10000
12500
15000
17500
20000
White
Black
Asian-Pac-Islander
Amer-Indian-Eskimo
Other
Race


Sex
Income
<=50K USD
>50K USD
count
14000
12000
10000
8000
6000
4000
2000
0
Male
Female
Sex


Native country
Income
<=50K USD
>50K USD
count
20000
15000
10000
5000
0
United-States
Cuba
Jamaica
India
Mexico
South
Puerto-Rico
Honduras
England
Canada
Germany
Iran
Philippines
Italy
Poland
Columbia
Cambodia
Thailand
Ecuador
Laos
Taiwan
Haiti
Portugal
Dominican-Republic
El-Salvador
France
Guatemala
China
Japan
Yugoslavia
Peru
Outlying-US(Guam-USVI-etc)
Scotland
Trinadad&Tobago
Greece
Nicaragua
Vietnam
Hong
Ireland
Hungary
Holand-Netherlands
Native country



Handling the NaN values

In [ ]:
```
#First drop the target column
Target=Data.pop('Income')
```

In [ ]:
```
Target.value_counts()
```

Out[ ]:
```
<=50K USD    24720
>50K USD      7841
Name: Income, dtype: int64
```

In [ ]:
```
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Age             32561 non-null  int64 
 1   Workclass       30725 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   Education       32561 non-null  object
 4   Education num   32561 non-null  int64 
 5   Marital status  32561 non-null  object
 6   Occupation      30718 non-null  object
 7   Relationship    32561 non-null  object
 8   Race            32561 non-null  object
 9   Sex             32561 non-null  object
 10  Capital gain    32561 non-null  int64 
 11  Capital loss    32561 non-null  int64 
 12  Hours per week  32561 non-null  int64 
 13  Native country  31978 non-null  object
dtypes: int64(6), object(8)
memory usage: 3.5+ MB
```

In [ ]:
```
#replacing the NA values with question mark once again for imputation purpose
Data.fillna('?',inplace=True)
```

In [ ]:
```
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Age             32561 non-null  int64 
 1   Workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   Education       32561 non-null  object
 4   Education num   32561 non-null  int64 
 5   Marital status  32561 non-null  object
 6   Occupation      32561 non-null  object
 7   Relationship    32561 non-null  object
 8   Race            32561 non-null  object
 9   Sex             32561 non-null  object
 10  Capital gain    32561 non-null  int64 
 11  Capital loss    32561 non-null  int64 
 12  Hours per week  32561 non-null  int64 
 13  Native country  32561 non-null  object
dtypes: int64(6), object(8)
memory usage: 3.5+ MB
```

In [ ]:
```
#Dropping the categorical columns hacving the question mark
Cat_miss_cols=['Workclass','Occupation','Native country']
Dropped_cat_col=Data[Cat_miss_cols]
Data.drop(Cat_miss_cols,axis=1,inplace=True)
```

In [ ]:
```
Data.info()#check the data again before fitting the K protos
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Age             32561 non-null  int64 
 1   fnlwgt          32561 non-null  int64 
 2   Education       32561 non-null  object
 3   Education num   32561 non-null  int64 
 4   Marital status  32561 non-null  object
 5   Relationship    32561 non-null  object
 6   Race            32561 non-null  object
 7   Sex             32561 non-null  object
 8   Capital gain    32561 non-null  int64 
 9   Capital loss    32561 non-null  int64 
 10  Hours per week  32561 non-null  int64 
dtypes: int64(6), object(5)
memory usage: 2.7+ MB
```

In [ ]:
```
kproto = KPrototypes(n_clusters=4, init='Cao', verbose=1)
# Fit the K-Prototypes model using the entire dataset except the target and the mis
clusters = kproto.fit_predict(Data,categorical=[2,4,5,6,7])
```

```
Initialization method and algorithm are deterministic. Setting n_init to 1.
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run: 1, iteration: 1/100, moves: 6382, ncost: 64918174760093.59
Run: 1, iteration: 2/100, moves: 2679, ncost: 62719236483521.266
Run: 1, iteration: 3/100, moves: 1786, ncost: 61480641233710.39
Run: 1, iteration: 4/100, moves: 1984, ncost: 60251922044607.9
Run: 1, iteration: 5/100, moves: 2062, ncost: 58947251425035.22
Run: 1, iteration: 6/100, moves: 1917, ncost: 57772264989359.875
Run: 1, iteration: 7/100, moves: 1572, ncost: 56769835569756.09
Run: 1, iteration: 8/100, moves: 1266, ncost: 55793706249941.055
Run: 1, iteration: 9/100, moves: 1005, ncost: 55045094200462.68
Run: 1, iteration: 10/100, moves: 680, ncost: 54696133846450.89
Run: 1, iteration: 11/100, moves: 575, ncost: 54457533579250.14
Run: 1, iteration: 12/100, moves: 479, ncost: 54230954355052.914
Run: 1, iteration: 13/100, moves: 400, ncost: 54105842998349.45
Run: 1, iteration: 14/100, moves: 275, ncost: 54027907669440.87
Run: 1, iteration: 15/100, moves: 153, ncost: 53993693260897.16
Run: 1, iteration: 16/100, moves: 102, ncost: 53980900800559.375
Run: 1, iteration: 17/100, moves: 91, ncost: 53975974714635.18
Run: 1, iteration: 18/100, moves: 62, ncost: 53973604531121.19
Run: 1, iteration: 19/100, moves: 50, ncost: 53972239125118.586
Run: 1, iteration: 20/100, moves: 36, ncost: 53971736750754.375
Run: 1, iteration: 21/100, moves: 27, ncost: 53971313039330.76
Run: 1, iteration: 22/100, moves: 22, ncost: 53971084145749.79
Run: 1, iteration: 23/100, moves: 9, ncost: 53971045914847.02
Run: 1, iteration: 24/100, moves: 2, ncost: 53971042517570.336
Run: 1, iteration: 25/100, moves: 0, ncost: 53971042517570.336
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run: 2, iteration: 1/100, moves: 3661, ncost: 54756262961655.72
Run: 2, iteration: 2/100, moves: 1505, ncost: 54044574726271.15
Run: 2, iteration: 3/100, moves: 458, ncost: 53980116388642.38
Run: 2, iteration: 4/100, moves: 170, ncost: 53972078786950.734
Run: 2, iteration: 5/100, moves: 60, ncost: 53969725178134.92
Run: 2, iteration: 6/100, moves: 18, ncost: 53969243053395.164
Run: 2, iteration: 7/100, moves: 2, ncost: 53969242182858.586
Run: 2, iteration: 8/100, moves: 0, ncost: 53969242182858.586
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run: 3, iteration: 1/100, moves: 12144, ncost: 65724363422962.41
Run: 3, iteration: 2/100, moves: 6080, ncost: 55544537143723.945
Run: 3, iteration: 3/100, moves: 1991, ncost: 54186162914735.46
Run: 3, iteration: 4/100, moves: 582, ncost: 54010367034055.19
Run: 3, iteration: 5/100, moves: 186, ncost: 53982400689867.72
Run: 3, iteration: 6/100, moves: 91, ncost: 53974273856472.02
Run: 3, iteration: 7/100, moves: 32, ncost: 53972562579693.72
Run: 3, iteration: 8/100, moves: 29, ncost: 53971857233935.99
Run: 3, iteration: 9/100, moves: 28, ncost: 53971374986890.97
Run: 3, iteration: 10/100, moves: 21, ncost: 53971119460821.92
Run: 3, iteration: 11/100, moves: 11, ncost: 53971045914847.02
Run: 3, iteration: 12/100, moves: 2, ncost: 53971042517570.336
Run: 3, iteration: 13/100, moves: 0, ncost: 53971042517570.336
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run: 4, iteration: 1/100, moves: 2392, ncost: 59310381722677.99
Run: 4, iteration: 2/100, moves: 2236, ncost: 57776814169773.59
Run: 4, iteration: 3/100, moves: 1896, ncost: 56543930254535.84
Run: 4, iteration: 4/100, moves: 1469, ncost: 55521137410575.984
Run: 4, iteration: 5/100, moves: 988, ncost: 54918450736532.51
```

```
Run: 4, iteration: 6/100, moves: 640, ncost: 54633568475379.3
Run: 4, iteration: 7/100, moves: 543, ncost: 54401231157732.72
Run: 4, iteration: 8/100, moves: 464, ncost: 54190702141800.25
Run: 4, iteration: 9/100, moves: 349, ncost: 54089247444634.08
Run: 4, iteration: 10/100, moves: 253, ncost: 54017876679150.82
Run: 4, iteration: 11/100, moves: 130, ncost: 53991637489854.02
Run: 4, iteration: 12/100, moves: 109, ncost: 53979660452002.83
Run: 4, iteration: 13/100, moves: 94, ncost: 53974985708531.93
Run: 4, iteration: 14/100, moves: 60, ncost: 53972962329291.625
Run: 4, iteration: 15/100, moves: 42, ncost: 53972053973687.29
Run: 4, iteration: 16/100, moves: 29, ncost: 53971666936766.625
Run: 4, iteration: 17/100, moves: 25, ncost: 53971277653693.5
Run: 4, iteration: 18/100, moves: 19, ncost: 53971081196988.13
Run: 4, iteration: 19/100, moves: 8, ncost: 53971045914847.02
Run: 4, iteration: 20/100, moves: 2, ncost: 53971042517570.336
Run: 4, iteration: 21/100, moves: 0, ncost: 53971042517570.336
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run: 5, iteration: 1/100, moves: 6045, ncost: 61449088530937.7
Run: 5, iteration: 2/100, moves: 1963, ncost: 60196045817527.516
Run: 5, iteration: 3/100, moves: 2138, ncost: 58883366248237.81
Run: 5, iteration: 4/100, moves: 1982, ncost: 57691392596928.2
Run: 5, iteration: 5/100, moves: 1585, ncost: 56664984040943.14
Run: 5, iteration: 6/100, moves: 1292, ncost: 55673553682511.84
Run: 5, iteration: 7/100, moves: 995, ncost: 54982358536379.53
Run: 5, iteration: 8/100, moves: 646, ncost: 54669986988417.19
Run: 5, iteration: 9/100, moves: 561, ncost: 54432458042542.82
Run: 5, iteration: 10/100, moves: 473, ncost: 54213042842259.32
Run: 5, iteration: 11/100, moves: 367, ncost: 54101070931683.42
Run: 5, iteration: 12/100, moves: 278, ncost: 54022047239783.3
Run: 5, iteration: 13/100, moves: 143, ncost: 53992032752264.83
Run: 5, iteration: 14/100, moves: 110, ncost: 53979771441098.4
Run: 5, iteration: 15/100, moves: 96, ncost: 53974985708531.93
Run: 5, iteration: 16/100, moves: 60, ncost: 53972962329291.625
Run: 5, iteration: 17/100, moves: 42, ncost: 53972053973687.29
Run: 5, iteration: 18/100, moves: 29, ncost: 53971666936766.625
Run: 5, iteration: 19/100, moves: 25, ncost: 53971277653693.5
Run: 5, iteration: 20/100, moves: 19, ncost: 53971081196988.13
Run: 5, iteration: 21/100, moves: 8, ncost: 53971045914847.02
Run: 5, iteration: 22/100, moves: 2, ncost: 53971042517570.336
Run: 5, iteration: 23/100, moves: 0, ncost: 53971042517570.336
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run: 6, iteration: 1/100, moves: 4350, ncost: 63416714231528.41
Run: 6, iteration: 2/100, moves: 1478, ncost: 62586094513285.92
Run: 6, iteration: 3/100, moves: 1820, ncost: 61655067066168.02
Run: 6, iteration: 4/100, moves: 1991, ncost: 60517592817145.164
Run: 6, iteration: 5/100, moves: 2070, ncost: 59246418971640.12
Run: 6, iteration: 6/100, moves: 1999, ncost: 58006251340208.97
Run: 6, iteration: 7/100, moves: 1629, ncost: 57004448571629.04
Run: 6, iteration: 8/100, moves: 1372, ncost: 55974999177956.89
Run: 6, iteration: 9/100, moves: 1107, ncost: 55146889843304.92
Run: 6, iteration: 10/100, moves: 748, ncost: 54739381303693.07
Run: 6, iteration: 11/100, moves: 583, ncost: 54500486086702.69
Run: 6, iteration: 12/100, moves: 504, ncost: 54258476718628.18
Run: 6, iteration: 13/100, moves: 403, ncost: 54120984050781.93
Run: 6, iteration: 14/100, moves: 298, ncost: 54035549312018.33
Run: 6, iteration: 15/100, moves: 181, ncost: 53994576123829.06
Run: 6, iteration: 16/100, moves: 106, ncost: 53981355670386.805
Run: 6, iteration: 17/100, moves: 91, ncost: 53976313808444.26
Run: 6, iteration: 18/100, moves: 67, ncost: 53973640142548.73
Run: 6, iteration: 19/100, moves: 52, ncost: 53972239125118.586
```

```
Run: 6, iteration: 20/100, moves: 36, ncost: 53971736750754.375
Run: 6, iteration: 21/100, moves: 27, ncost: 53971313039330.76
Run: 6, iteration: 22/100, moves: 22, ncost: 53971084145749.79
Run: 6, iteration: 23/100, moves: 9, ncost: 53971045914847.02
Run: 6, iteration: 24/100, moves: 2, ncost: 53971042517570.336
Run: 6, iteration: 25/100, moves: 0, ncost: 53971042517570.336
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run: 7, iteration: 1/100, moves: 6215, ncost: 61091457159411.86
Run: 7, iteration: 2/100, moves: 2652, ncost: 58138307232249.69
Run: 7, iteration: 3/100, moves: 1298, ncost: 57011774416055.91
Run: 7, iteration: 4/100, moves: 1126, ncost: 55964429356652.14
Run: 7, iteration: 5/100, moves: 1010, ncost: 55128056187127.25
Run: 7, iteration: 6/100, moves: 711, ncost: 54728151795779.914
Run: 7, iteration: 7/100, moves: 584, ncost: 54489114674453.164
Run: 7, iteration: 8/100, moves: 488, ncost: 54254811757968.98
Run: 7, iteration: 9/100, moves: 395, ncost: 54120861377380.97
Run: 7, iteration: 10/100, moves: 296, ncost: 54035549312018.33
Run: 7, iteration: 11/100, moves: 181, ncost: 53994576123829.06
Run: 7, iteration: 12/100, moves: 106, ncost: 53981355670386.805
Run: 7, iteration: 13/100, moves: 91, ncost: 53976313808444.26
Run: 7, iteration: 14/100, moves: 67, ncost: 53973640142548.73
Run: 7, iteration: 15/100, moves: 52, ncost: 53972239125118.586
Run: 7, iteration: 16/100, moves: 36, ncost: 53971736750754.375
Run: 7, iteration: 17/100, moves: 27, ncost: 53971313039330.76
Run: 7, iteration: 18/100, moves: 22, ncost: 53971084145749.79
Run: 7, iteration: 19/100, moves: 9, ncost: 53971045914847.02
Run: 7, iteration: 20/100, moves: 2, ncost: 53971042517570.336
Run: 7, iteration: 21/100, moves: 0, ncost: 53971042517570.336
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run: 8, iteration: 1/100, moves: 4932, ncost: 68583225226868.04
Run: 8, iteration: 2/100, moves: 3224, ncost: 65762887859401.89
Run: 8, iteration: 3/100, moves: 2222, ncost: 64427199371019.73
Run: 8, iteration: 4/100, moves: 1781, ncost: 63556693037571.74
Run: 8, iteration: 5/100, moves: 1592, ncost: 62850888377626.48
Run: 8, iteration: 6/100, moves: 1718, ncost: 62036774941067.71
Run: 8, iteration: 7/100, moves: 1970, ncost: 60966190009192.42
Run: 8, iteration: 8/100, moves: 2097, ncost: 59701534737717.664
Run: 8, iteration: 9/100, moves: 2044, ncost: 58424347139962.266
Run: 8, iteration: 10/100, moves: 1803, ncost: 57353519714931.59
Run: 8, iteration: 11/100, moves: 1478, ncost: 56311012990130.95
Run: 8, iteration: 12/100, moves: 1188, ncost: 55396919736442.58
Run: 8, iteration: 13/100, moves: 839, ncost: 54873666642643.234
Run: 8, iteration: 14/100, moves: 607, ncost: 54604361699704.43
Run: 8, iteration: 15/100, moves: 534, ncost: 54361920912641.125
Run: 8, iteration: 16/100, moves: 431, ncost: 54177077650586.72
Run: 8, iteration: 17/100, moves: 315, ncost: 54083326768012.36
Run: 8, iteration: 18/100, moves: 239, ncost: 54017271498835.13
Run: 8, iteration: 19/100, moves: 123, ncost: 53991637489854.02
Run: 8, iteration: 20/100, moves: 109, ncost: 53979660452002.83
Run: 8, iteration: 21/100, moves: 94, ncost: 53974985708531.93
Run: 8, iteration: 22/100, moves: 60, ncost: 53972962329291.625
Run: 8, iteration: 23/100, moves: 42, ncost: 53972053973687.29
Run: 8, iteration: 24/100, moves: 29, ncost: 53971666936766.625
Run: 8, iteration: 25/100, moves: 25, ncost: 53971277653693.5
Run: 8, iteration: 26/100, moves: 19, ncost: 53971081196988.13
Run: 8, iteration: 27/100, moves: 8, ncost: 53971045914847.02
Run: 8, iteration: 28/100, moves: 2, ncost: 53971042517570.336
Run: 8, iteration: 29/100, moves: 0, ncost: 53971042517570.336
Init: initializing centroids
Init: initializing clusters
```

```
Starting iterations...
Run: 9, iteration: 1/100, moves: 2635, ncost: 66071435192432.87
Run: 9, iteration: 2/100, moves: 2253, ncost: 64604525031366.7
Run: 9, iteration: 3/100, moves: 1843, ncost: 63665650778123.76
Run: 9, iteration: 4/100, moves: 1594, ncost: 62963635741175.25
Run: 9, iteration: 5/100, moves: 1700, ncost: 62167988861813.23
Run: 9, iteration: 6/100, moves: 1907, ncost: 61148774198109.484
Run: 9, iteration: 7/100, moves: 2033, ncost: 59941539702582.016
Run: 9, iteration: 8/100, moves: 2140, ncost: 58605915887515.49
Run: 9, iteration: 9/100, moves: 1851, ncost: 57503493190430.92
Run: 9, iteration: 10/100, moves: 1561, ncost: 56455373005712.766
Run: 9, iteration: 11/100, moves: 1256, ncost: 55464833725451.6
Run: 9, iteration: 12/100, moves: 899, ncost: 54892139060989.375
Run: 9, iteration: 13/100, moves: 604, ncost: 54619849120441.33
Run: 9, iteration: 14/100, moves: 529, ncost: 54386619889166.48
Run: 9, iteration: 15/100, moves: 449, ncost: 54185485365789.055
Run: 9, iteration: 16/100, moves: 333, ncost: 54086758479431.63
Run: 9, iteration: 17/100, moves: 249, ncost: 54017271498835.13
Run: 9, iteration: 18/100, moves: 123, ncost: 53991637489854.02
Run: 9, iteration: 19/100, moves: 109, ncost: 53979660452002.83
Run: 9, iteration: 20/100, moves: 94, ncost: 53974985708531.93
Run: 9, iteration: 21/100, moves: 60, ncost: 53972962329291.625
Run: 9, iteration: 22/100, moves: 42, ncost: 53972053973687.29
Run: 9, iteration: 23/100, moves: 29, ncost: 53971666936766.625
Run: 9, iteration: 24/100, moves: 25, ncost: 53971277653693.5
Run: 9, iteration: 25/100, moves: 19, ncost: 53971081196988.13
Run: 9, iteration: 26/100, moves: 8, ncost: 53971045914847.02
Run: 9, iteration: 27/100, moves: 2, ncost: 53971042517570.336
Run: 9, iteration: 28/100, moves: 0, ncost: 53971042517570.336
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run: 10, iteration: 1/100, moves: 4997, ncost: 56615471732286.8
Run: 10, iteration: 2/100, moves: 2142, ncost: 54978700425861.54
Run: 10, iteration: 3/100, moves: 1078, ncost: 54586513113818.19
Run: 10, iteration: 4/100, moves: 688, ncost: 54319174974265.03
Run: 10, iteration: 5/100, moves: 490, ncost: 54148458310848.29
Run: 10, iteration: 6/100, moves: 337, ncost: 54056315770373.63
Run: 10, iteration: 7/100, moves: 215, ncost: 54003658971697.086
Run: 10, iteration: 8/100, moves: 105, ncost: 53987605228739.15
Run: 10, iteration: 9/100, moves: 100, ncost: 53978198428790.24
Run: 10, iteration: 10/100, moves: 85, ncost: 53974294876397.984
Run: 10, iteration: 11/100, moves: 57, ncost: 53972483253067.01
Run: 10, iteration: 12/100, moves: 35, ncost: 53971899110278.61
Run: 10, iteration: 13/100, moves: 33, ncost: 53971382182708.44
Run: 10, iteration: 14/100, moves: 25, ncost: 53971104694200.71
Run: 10, iteration: 15/100, moves: 13, ncost: 53971045914847.02
Run: 10, iteration: 16/100, moves: 2, ncost: 53971042517570.336
Run: 10, iteration: 17/100, moves: 0, ncost: 53971042517570.336
Best run was number 2
```

In [ ]:
```
Data['Cluster'] = clusters
```

In [ ]:
```
np.unique(clusters)
```

Out[ ]:
```
array([0, 1, 2, 3], dtype=uint16)
```

In [ ]:
```
Data[Cat_miss_cols]=Dropped_cat_col
```

In [ ]:
```
for cluster_id in range(4):  # Assuming 3 clusters
    cluster_data = Data[Data['Cluster'] == cluster_id]

    # Mode for each categorical variable
```

```
    cluster_modes = cluster_data.mode().iloc[0]

    # Impute missing values for the cluster
    for col in Cat_miss_cols:
       Data.loc[Data['Cluster'] == cluster_id,col]=Data[Data['Cluster'] == cluster_
```

In [ ]:
```
Data.drop('Cluster',axis=1,inplace=True)#Now drop the cluster column
```

In [ ]:
```
Num_cols=['Age','fnlwgt','Education num','Capital gain','Capital loss','Hours per w
Data[Num_cols]=Data[Num_cols].astype(str)#to perform ohe on numerical data we have
```

In [ ]:
```
Data#Recheck the data
```

Out[ ]:

	Age	fnlwgt	Education	Education num	Marital status	Relationship	Race	Sex	Capital gain	Capita loss
0	39	77516	Bachelors	13	Never-married	Not-in-family	White	Male	2174	(
1	50	83311	Bachelors	13	Married-civ-spouse	Husband	White	Male	0	(
2	38	215646	HS-grad	9	Divorced	Not-in-family	White	Male	0	(
3	53	234721	11th	7	Married-civ-spouse	Husband	Black	Male	0	(
4	28	338409	Bachelors	13	Married-civ-spouse	Wife	Black	Female	0	(
...	...	...	...	...	...	...	...	...	...	..
32556	27	257302	Assoc-acdm	12	Married-civ-spouse	Wife	White	Female	0	(
32557	40	154374	HS-grad	9	Married-civ-spouse	Husband	White	Male	0	(
32558	58	151910	HS-grad	9	Widowed	Unmarried	White	Female	0	(
32559	22	201490	HS-grad	9	Never-married	Own-child	White	Male	0	(
32560	52	287927	HS-grad	9	Married-civ-spouse	Wife	White	Female	15024	(

32561 rows × 14 columns

In [ ]:
```
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Age             32561 non-null  object
 1   fnlwgt          32561 non-null  object
 2   Education       32561 non-null  object
 3   Education num   32561 non-null  object
 4   Marital status  32561 non-null  object
 5   Relationship    32561 non-null  object
 6   Race            32561 non-null  object
 7   Sex             32561 non-null  object
 8   Capital gain    32561 non-null  object
 9   Capital loss    32561 non-null  object
 10  Hours per week  32561 non-null  object
 11  Workclass       32561 non-null  object
 12  Occupation      32561 non-null  object
 13  Native country  32561 non-null  object
dtypes: object(14)
memory usage: 3.5+ MB
```

In [ ]:
```
OHE_data=pd.get_dummies(Data,prefix_sep='*')#Perform One hot encoding on the data
OHE_target=pd.get_dummies(Target,drop_first=True)
```

In [ ]:
```
#OHE_data.columns#Reviewing the newly formed columns
```

In [ ]:
```
#Perform train test split with split size 0.25 for test
X_train,X_test,y_train,y_test=train_test_split(Data,Target,test_size=0.35,random_st
```

In [ ]:
```
np.unique(y_train, return_counts=True)
```

Out[ ]:
```
(array(['<=50K USD', '>50K USD'], dtype=object), array([16068,  5096]))
```

In [ ]:
```
def pipeline(x_train,y_train,x_test,y_test):
  est=[('com',ComplementNB()),('bern',BernoulliNB()),('cat',CategoricalNB())]
  vote=VotingClassifier(estimators=est,voting='soft')
  pipe=Pipeline([('ohe',OneHotEncoder(handle_unknown="ignore",sparse_output=False))
  skfold=StratifiedKFold(n_splits=10,shuffle=True,random_state=11)
  print("Training cv score: ",cross_val_score(pipe, x_train, y_train, cv=skfold).me
  pipe.fit(x_train,y_train)
  yhat_test=pipe.predict(x_test)
  yhat_prob=pipe.predict_proba(x_test)[:,1]
  return yhat_test,yhat_prob
```

In [ ]:
```
Output=pipeline(X_train.values,y_train,X_test.values,y_test)
yhat_test=Output[0]
yhat_test_prob=Output[1]
```

```
Training cv score:  0.8441692643850797
```

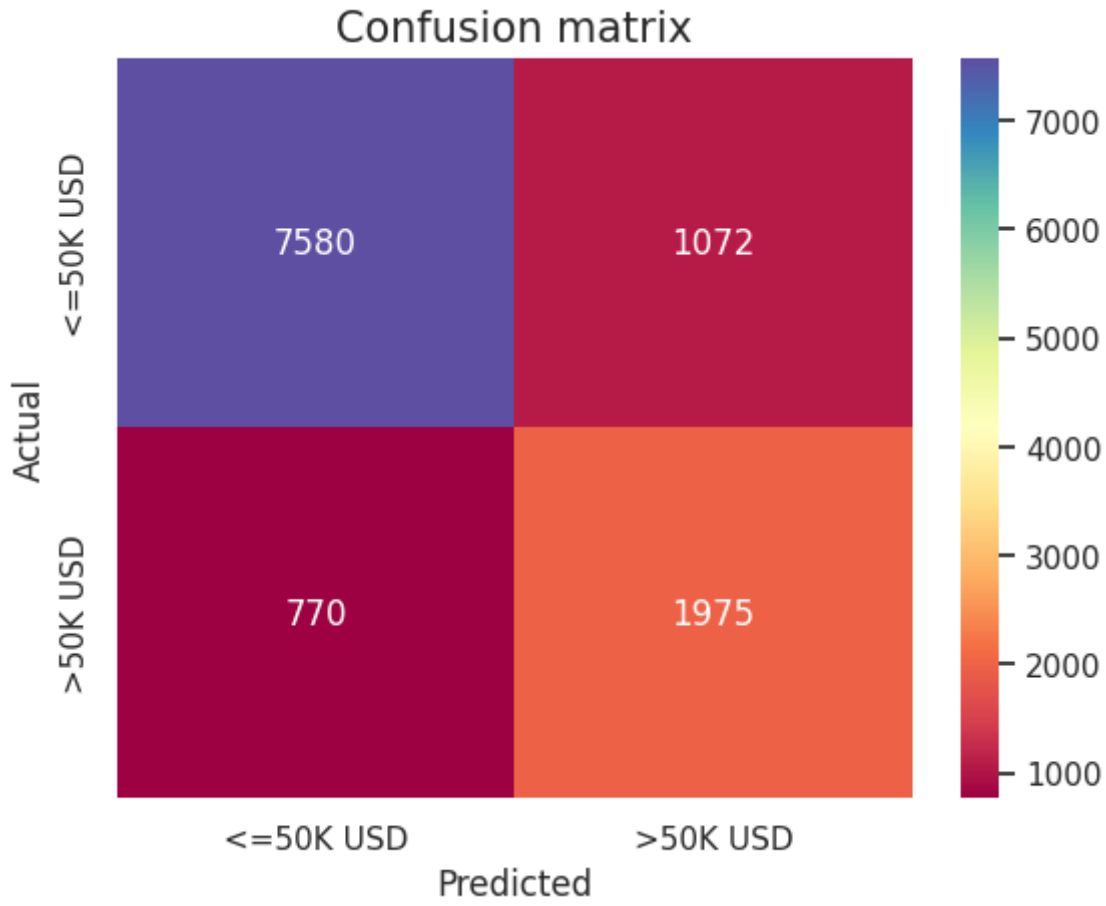
Model Evaluation

In [ ]:
```
#Test set report
print(classification_report(y_test,yhat_test))
```

	precision	recall	f1-score	support
<=50K USD	0.91	0.88	0.89	8652
>50K USD	0.65	0.72	0.68	2745
accuracy			0.84	11397
macro avg	0.78	0.80	0.79	11397
weighted avg	0.85	0.84	0.84	11397

Check the confusion matrix in the test set

In [ ]:
```
cm = confusion_matrix(y_test, yhat_test)
sns.heatmap(cm,annot=True,fmt='d',cmap='Spectral')
plt.title("Confusion matrix",fontsize=15)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.xticks(ticks=[0.5,1.5],labels=['<=50K USD','>50K USD'])
plt.yticks(ticks=[0.5,1.5],labels=['<=50K USD','>50K USD'])
plt.show()
```



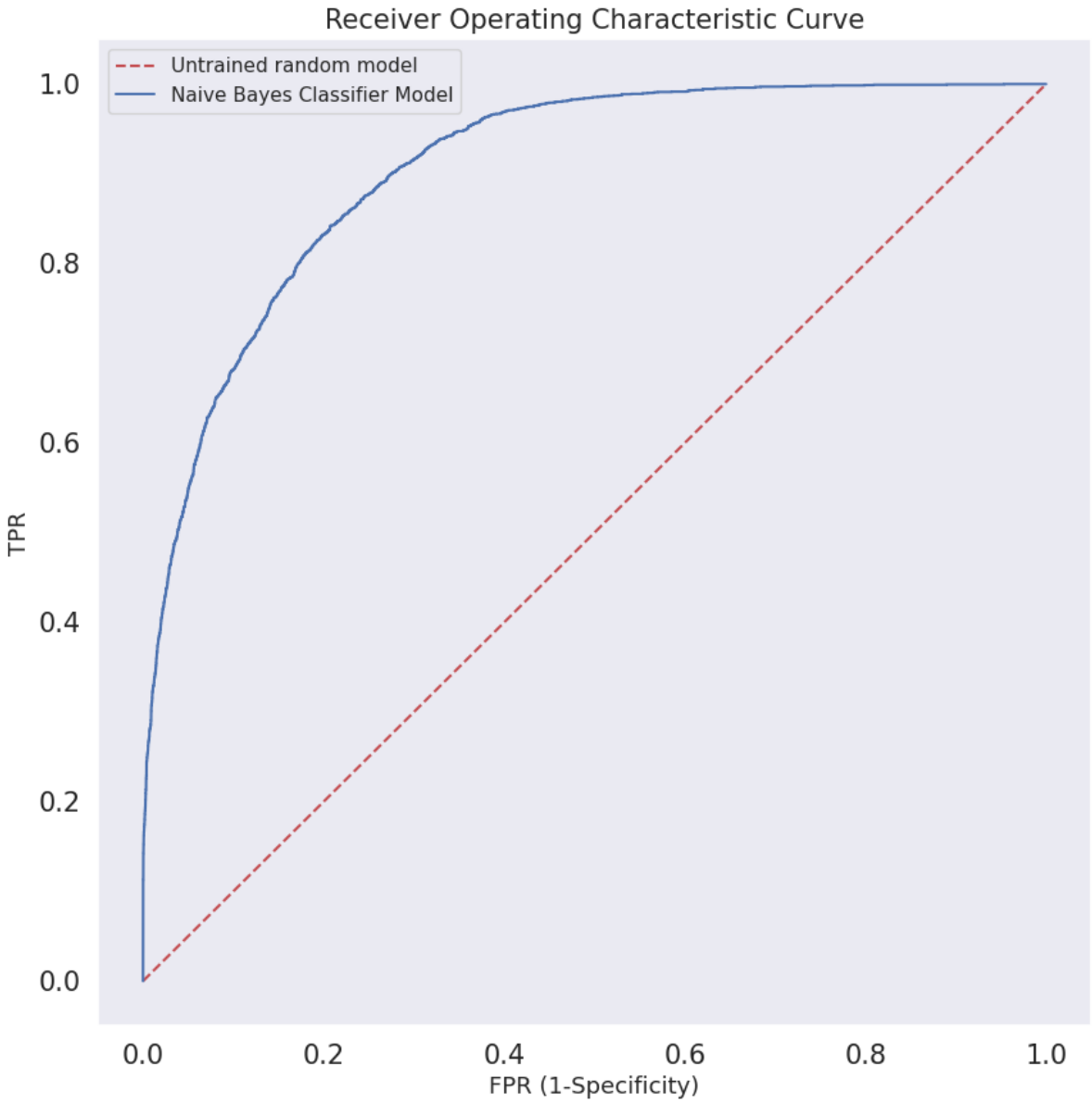
In [ ]:
```
y_test_num=[1 if i=='>50K USD' else 0 for i in y_test]
y_test_hat_num=[1 if i=='>50K USD' else 0 for i in yhat_test]
```

In [ ]:
```
fpr,tpr,threshold=roc_curve(y_test_num,yhat_test_prob)
print("Area of ROC for test set is",auc(fpr,tpr))# area under ROC curve
```

```
Area of ROC for test set is 0.9063491010848961
```

In [ ]:
```
#Plot the roc
plt.figure(figsize=(10,10))
plt.plot([0,1],[0,1],'r--',label='Untrained random model')
plt.plot(fpr,tpr,label='Naive Bayes Classifier Model')
```

```
plt.title("Receiver Operating Characteristic Curve",fontsize=15)
plt.ylabel("TPR",fontsize=13)
plt.xlabel("FPR (1-Specificity)",fontsize=13)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend()
plt.grid()
plt.show()
```



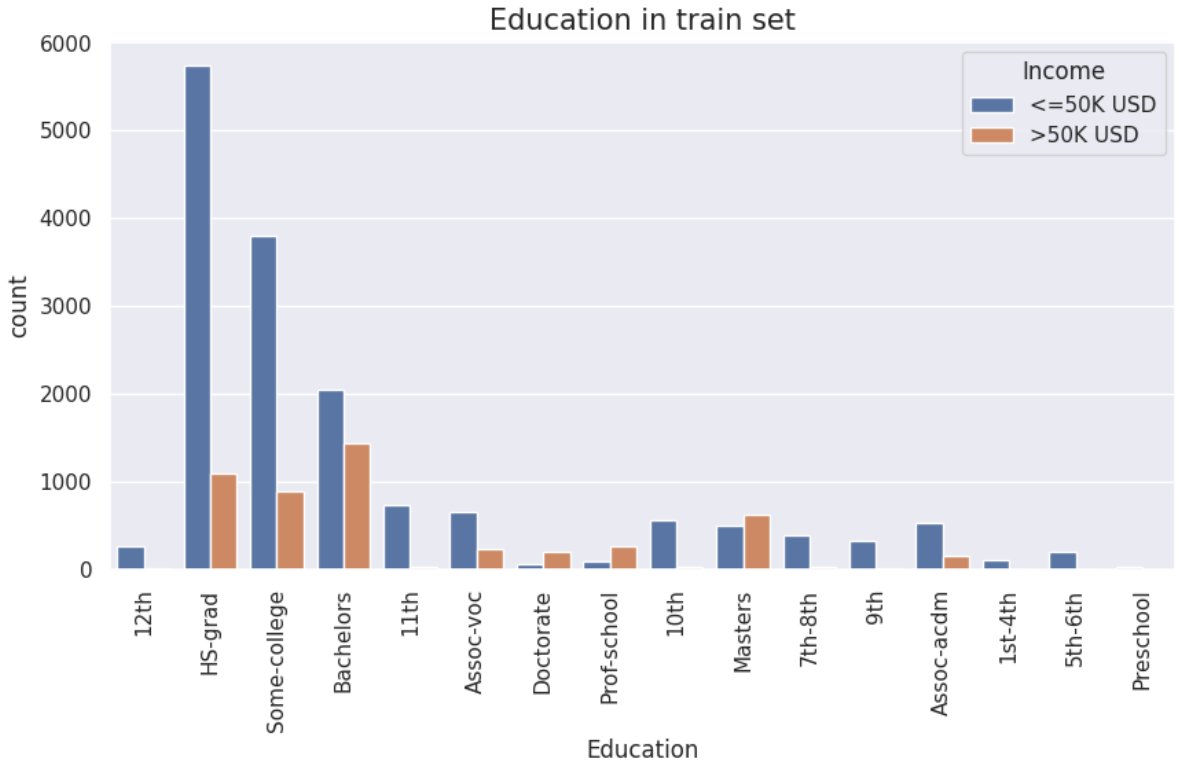
In [ ]:
```
X_test[Num_cols]=X_test[Num_cols].astype('float64')
X_train[Num_cols]=X_train[Num_cols].astype('float64')
```

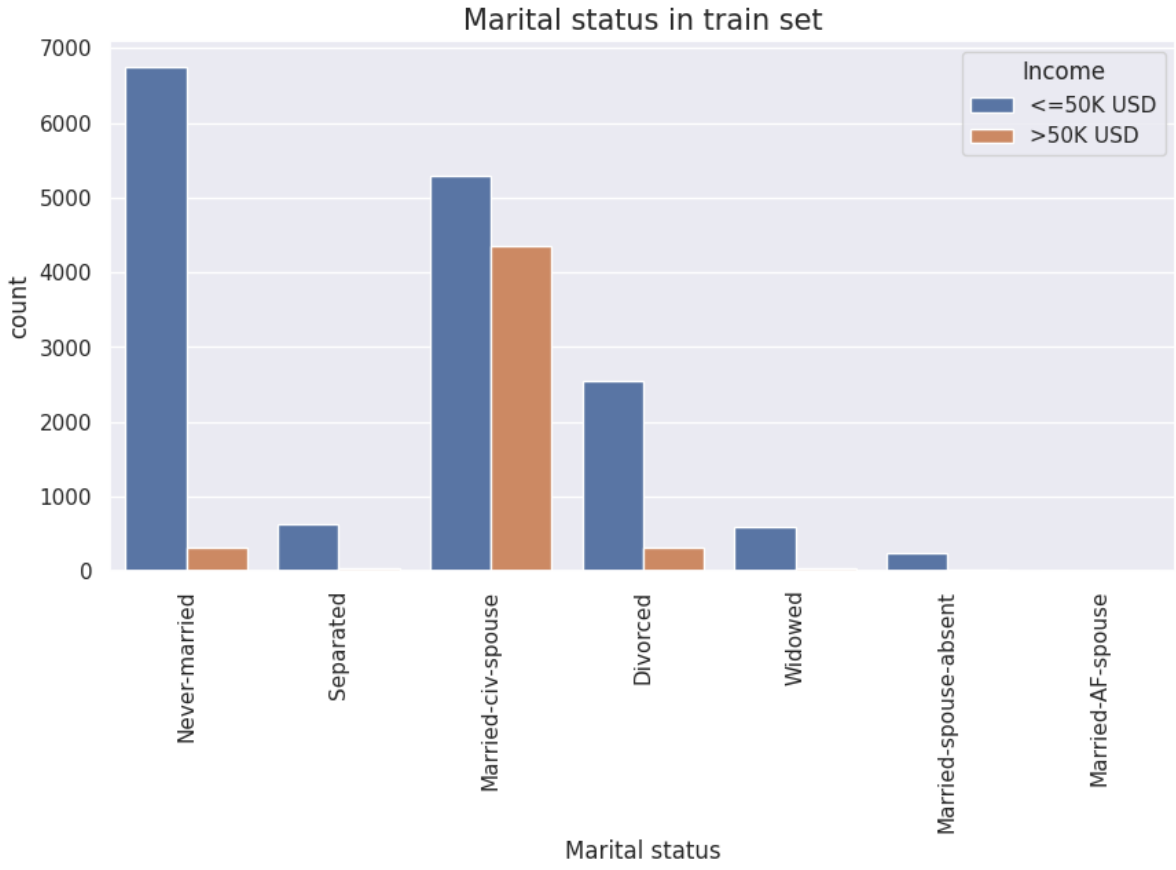
In [ ]:
```
X_test['Predicted Income']=yhat_test
X_train['Income']=y_train
```

Plot the distribution for trainset

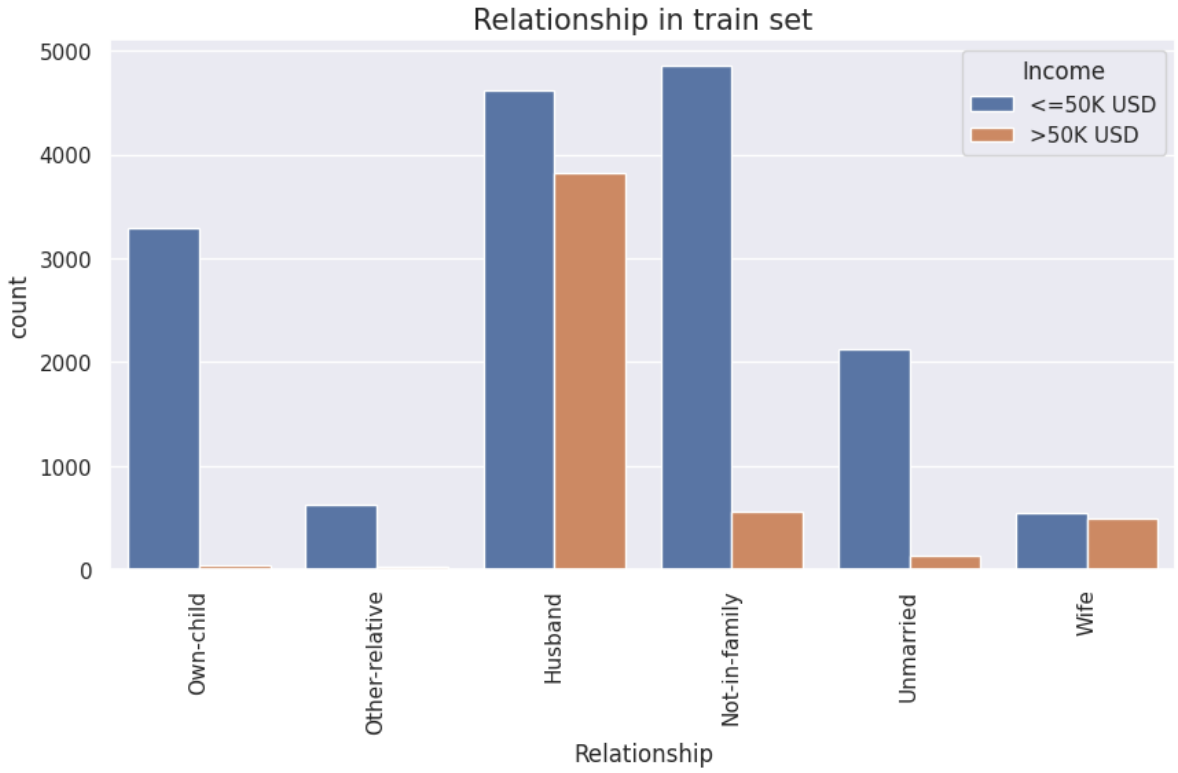
In [ ]:
```
sns.set()
for cat in Cat_col:
  plt.figure(figsize=(10,5))
  sns.countplot(x=cat,data=X_train,hue='Income',hue_order=['<=50K USD','>50K USD'])
  plt.title(f"{cat} in train set",fontsize=15)
  plt.xticks(rotation=90)
```

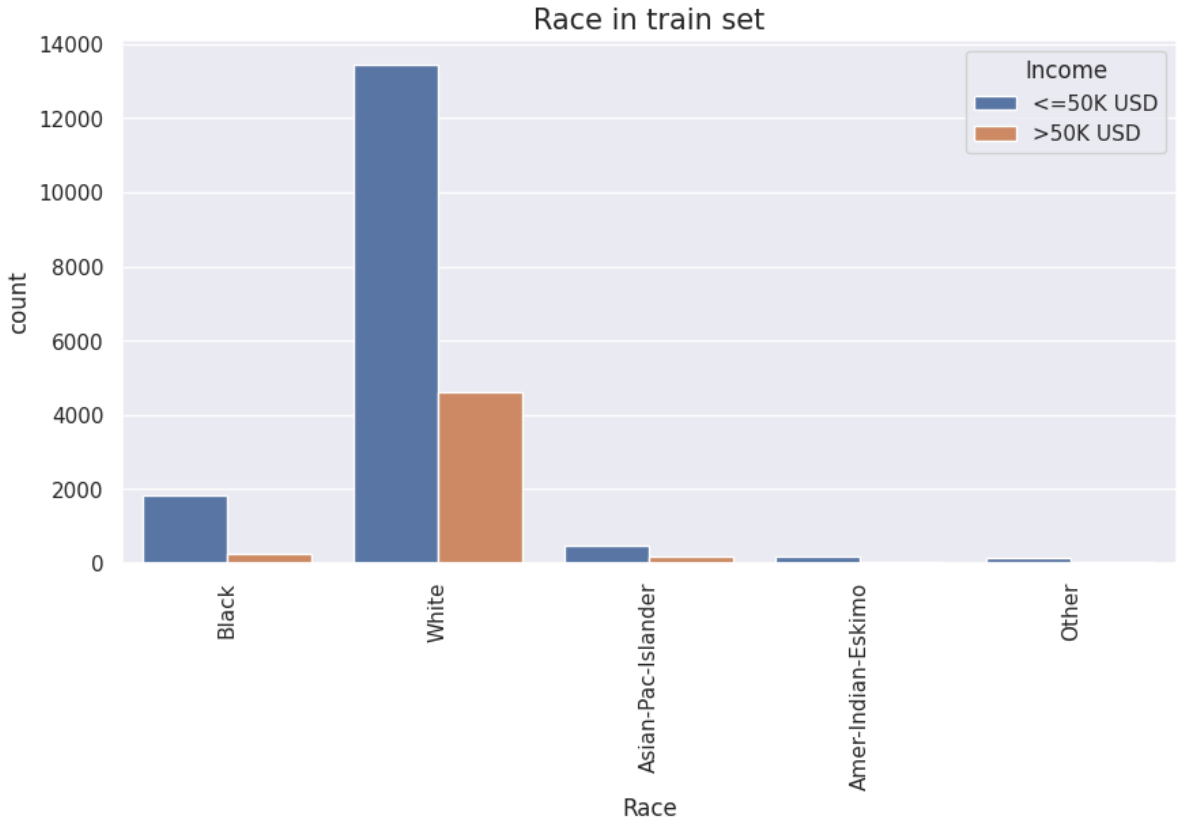
```
    plt.savefig(f"{cat} in train.png")
    plt.show()
```

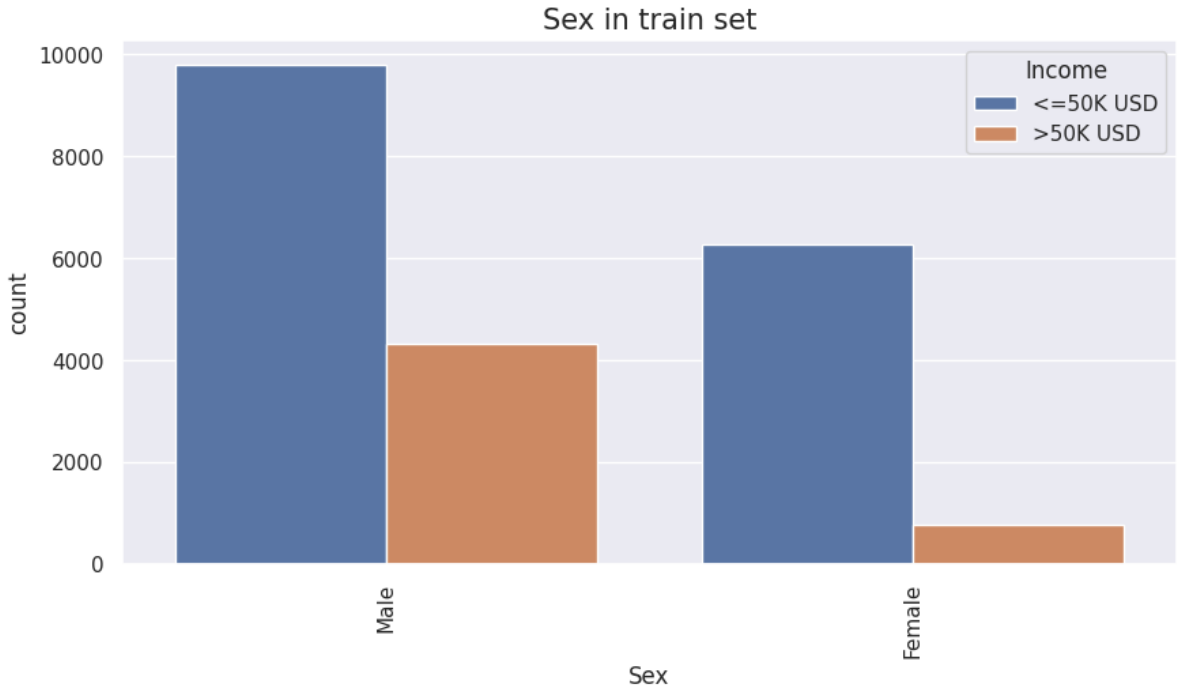


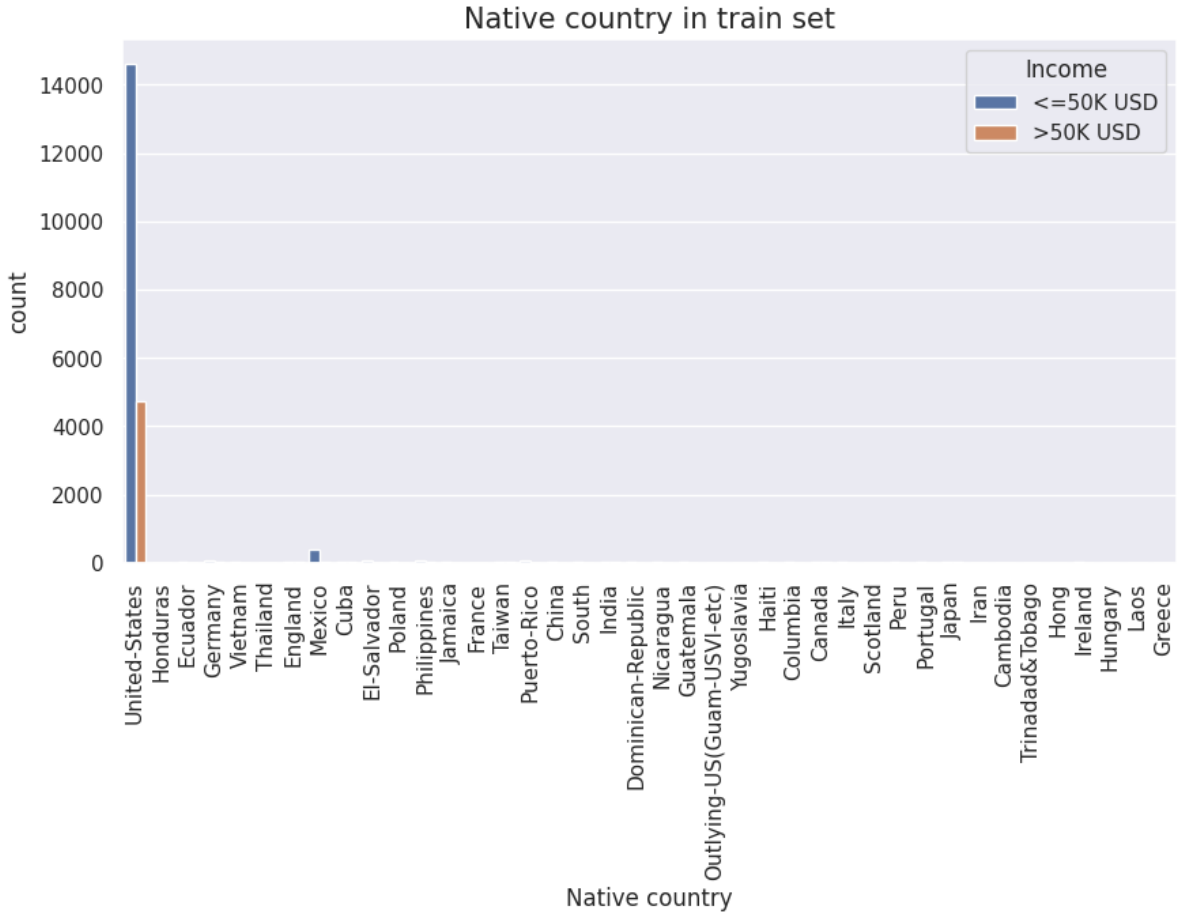

Marital status in train set
Income
<=50K USD
>50K USD
count
7000
6000
5000
4000
3000
2000
1000
0
Never-married
Separated
Married-civ-spouse
Divorced
Widowed
Married-spouse-absent
Married-AF-spouse
Marital status

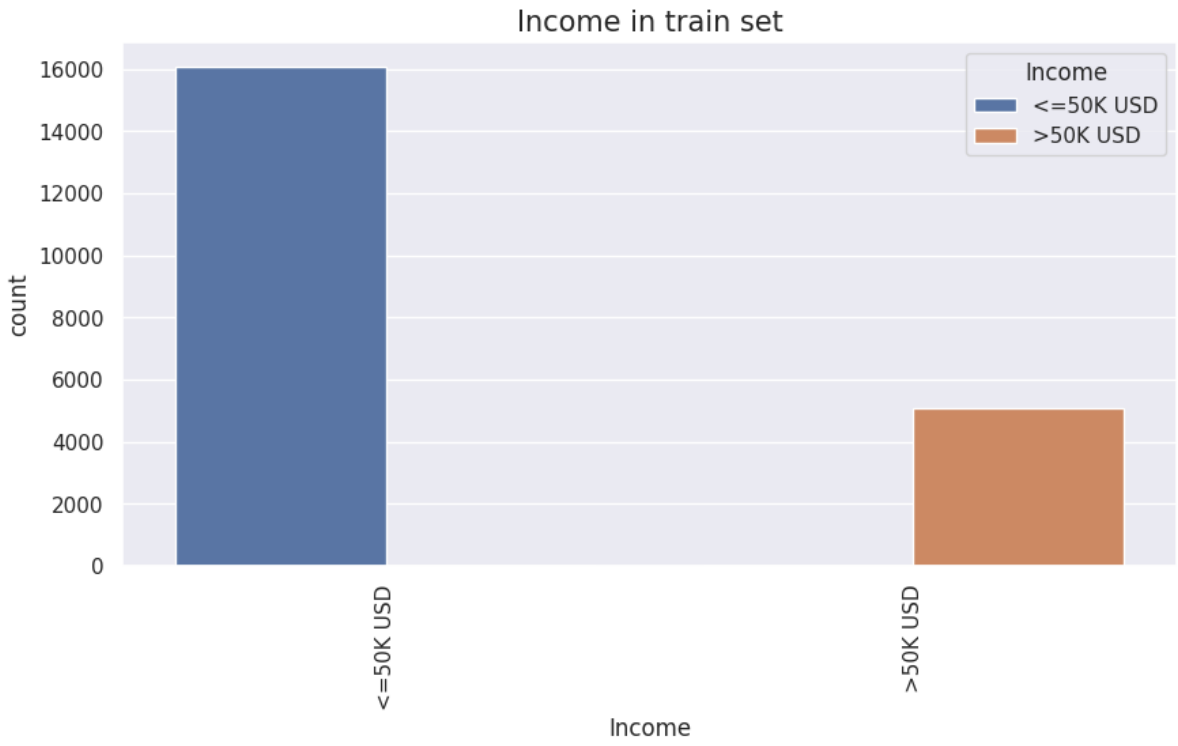

Occupation in train set
Income
<=50K USD
>50K USD
count
2500
2000
1500
1000
500
0
Farming-fishing
Other-service
Exec-managerial
Sales
Adm-clerical
Craft-repair
Prof-specialty
Tech-support
Handlers-cleaners
Machine-op-inspct
Protective-serv
Transport-moving
Priv-house-serv
Armed-Forces
Occupation


Relationship in train set
Income
<=50K USD
>50K USD
count
5000
4000
3000
2000
1000
0
Own-child
Other-relative
Husband
Not-in-family
Unmarried
Wife
Relationship


Race in train set
Income
<=50K USD
>50K USD
count
14000
12000
10000
8000
6000
4000
2000
0
Black
White
Asian-Pac-Islander
Amer-Indian-Eskimo
Other
Race

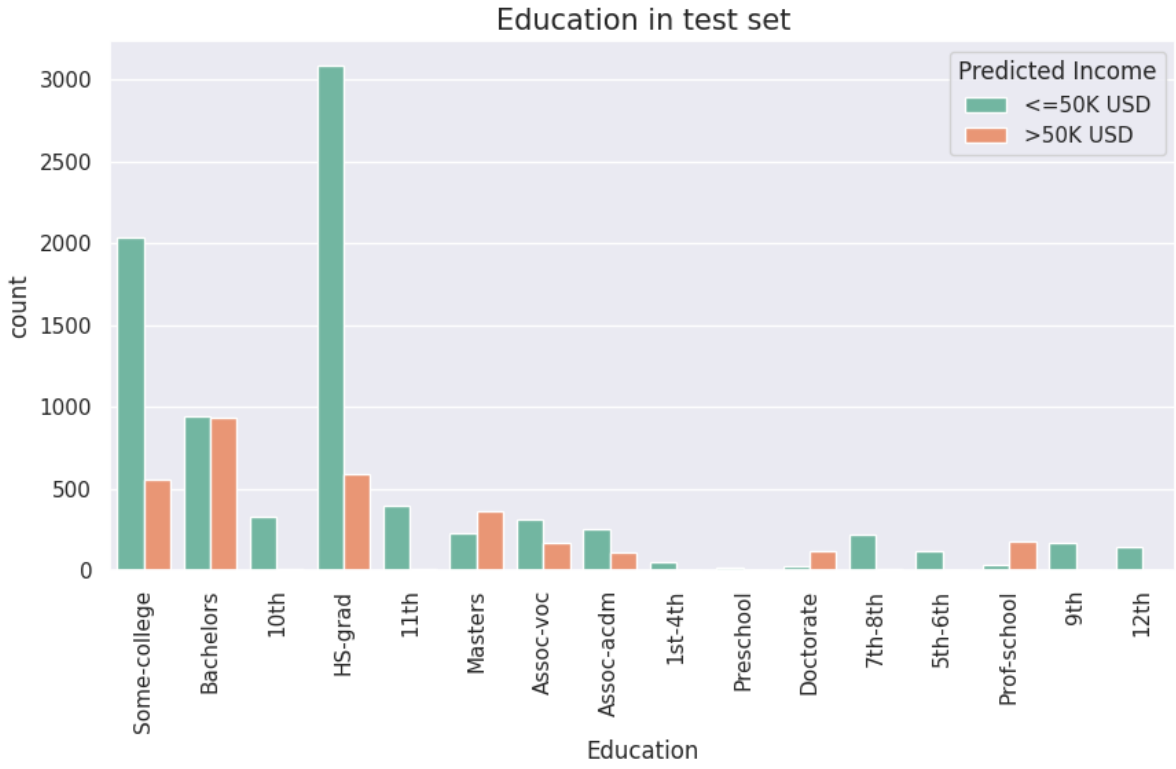

Sex in train set
Income
<=50K USD
>50K USD
count
10000
8000
6000
4000
2000
0
Male
Female
Sex

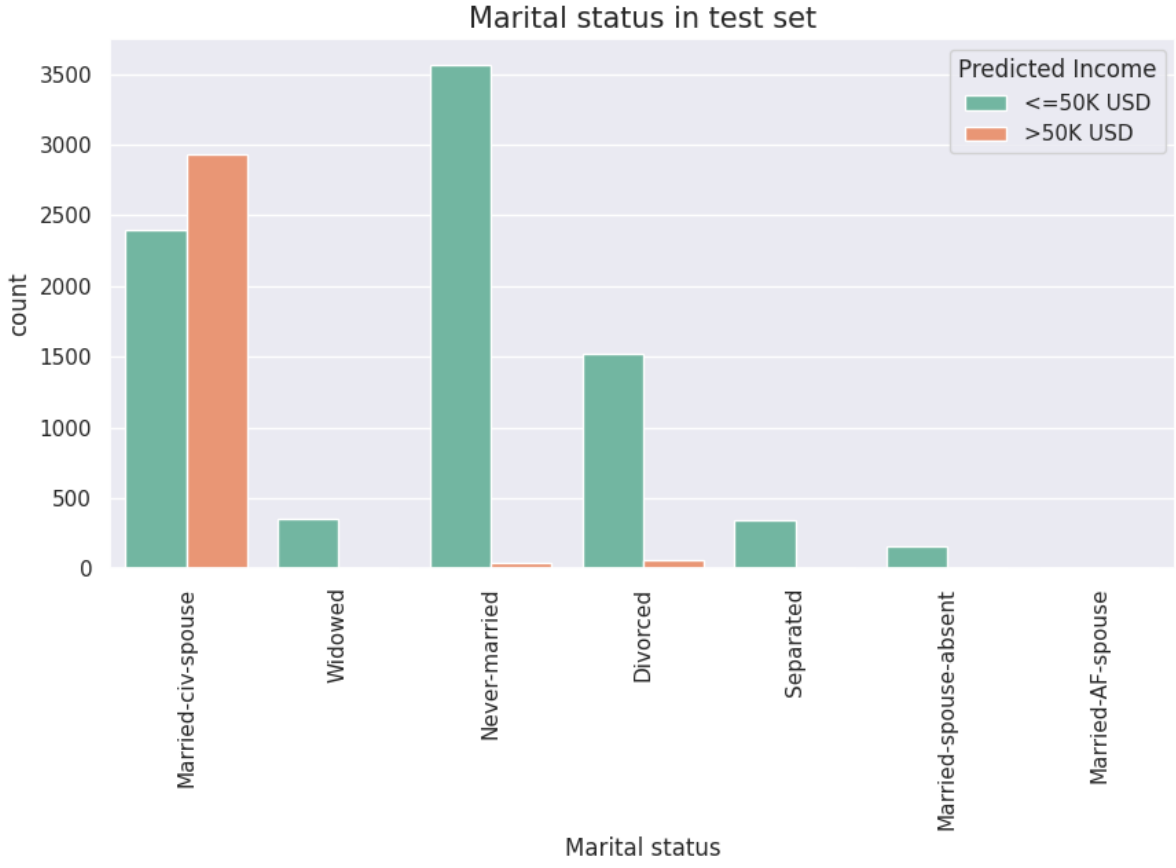

Native country in train set
Income
<=50K USD
>50K USD
count
14000
12000
10000
8000
6000
4000
2000
0
United-States
Honduras
Ecuador
Germany
Vietnam
Thailand
England
Mexico
Cuba
El-Salvador
Poland
Philippines
Jamaica
France
Taiwan
Puerto-Rico
China
South
India
Dominican-Republic
Nicaragua
Guatemala
Outlying-US(Guam-USVI-etc)
Yugoslavia
Haiti
Columbia
Canada
Italy
Scotland
Peru
Portugal
Japan
Iran
Cambodia
Trinadad&Tobago
Hong
Ireland
Hungary
Laos
Greece
Native country



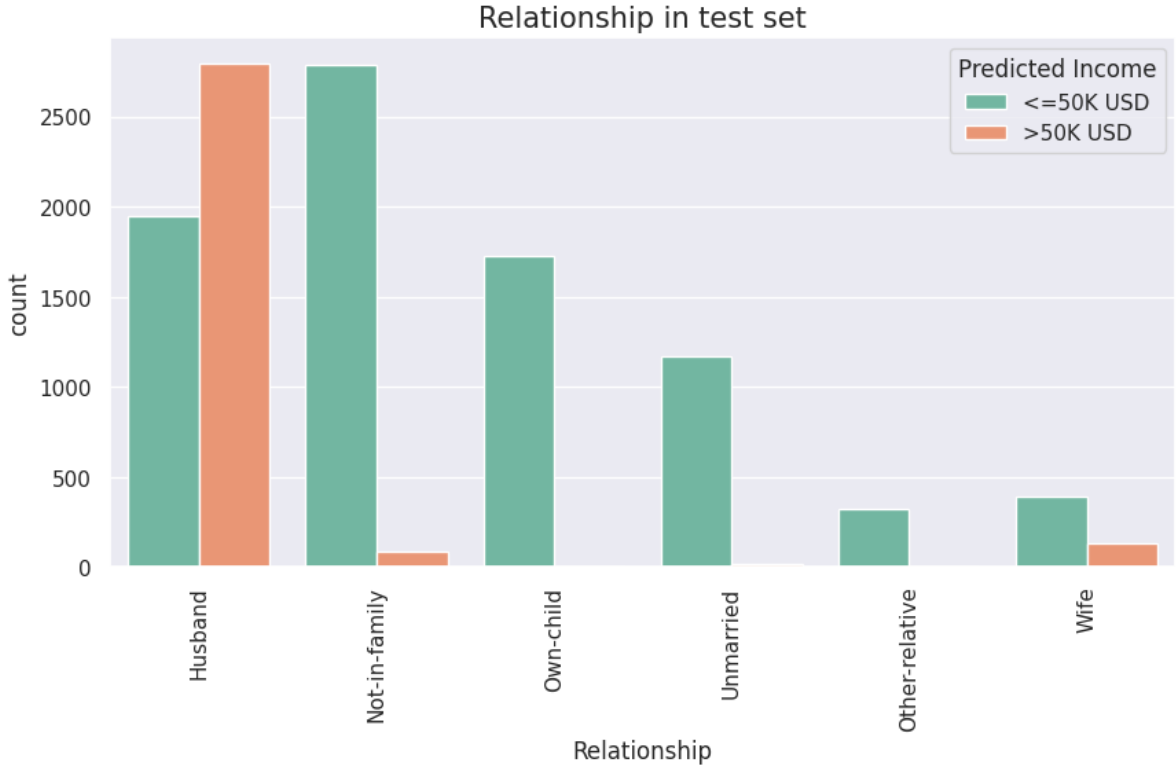
Plot the distribution for the predicted test set

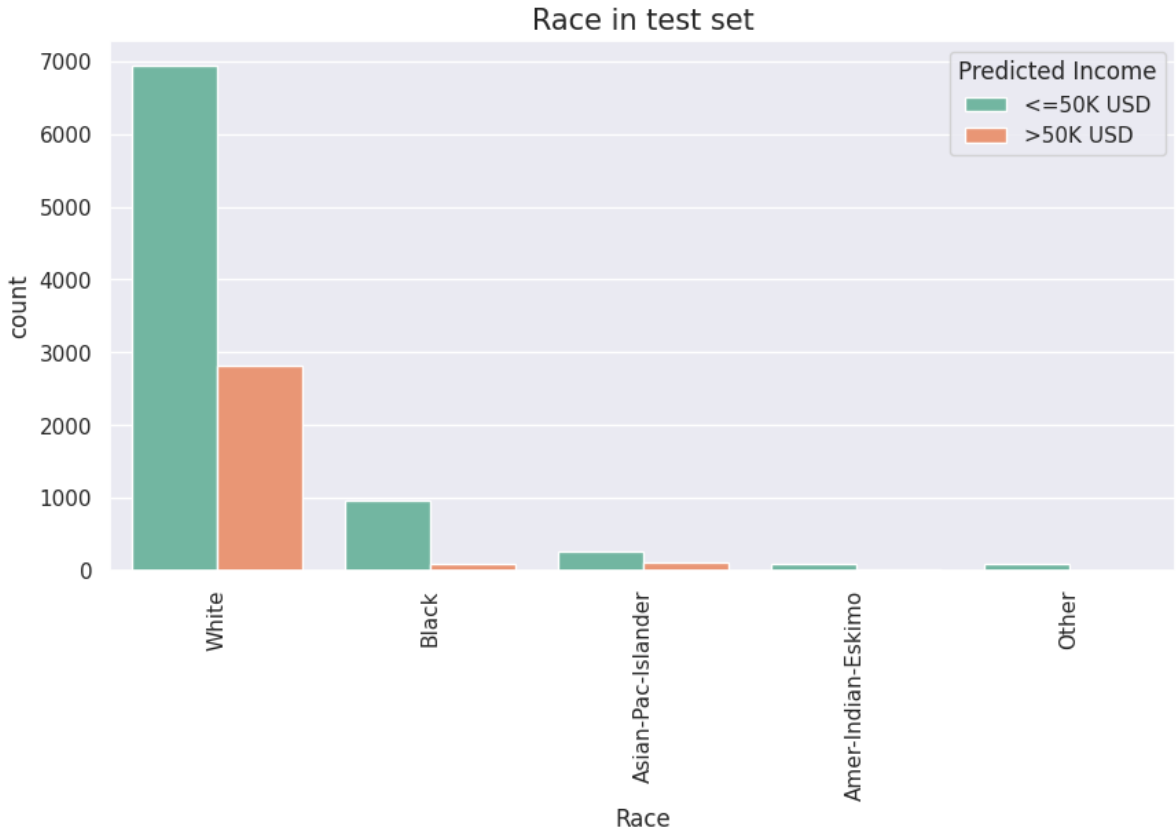
In [ ]:
```
sns.set()
cat_col=['Workclass', 'Education', 'Marital status', 'Occupation', 'Relationship',
for cat in cat_col:
  plt.figure(figsize=(10,5))
  sns.countplot(x=cat,data=X_test,hue='Predicted Income',hue_order=['<=50K USD','>5
  plt.title(f"{cat} in test set",fontsize=15)
  plt.xticks(rotation=90)
  plt.savefig(f"{cat} in test.png")
  plt.show()
```

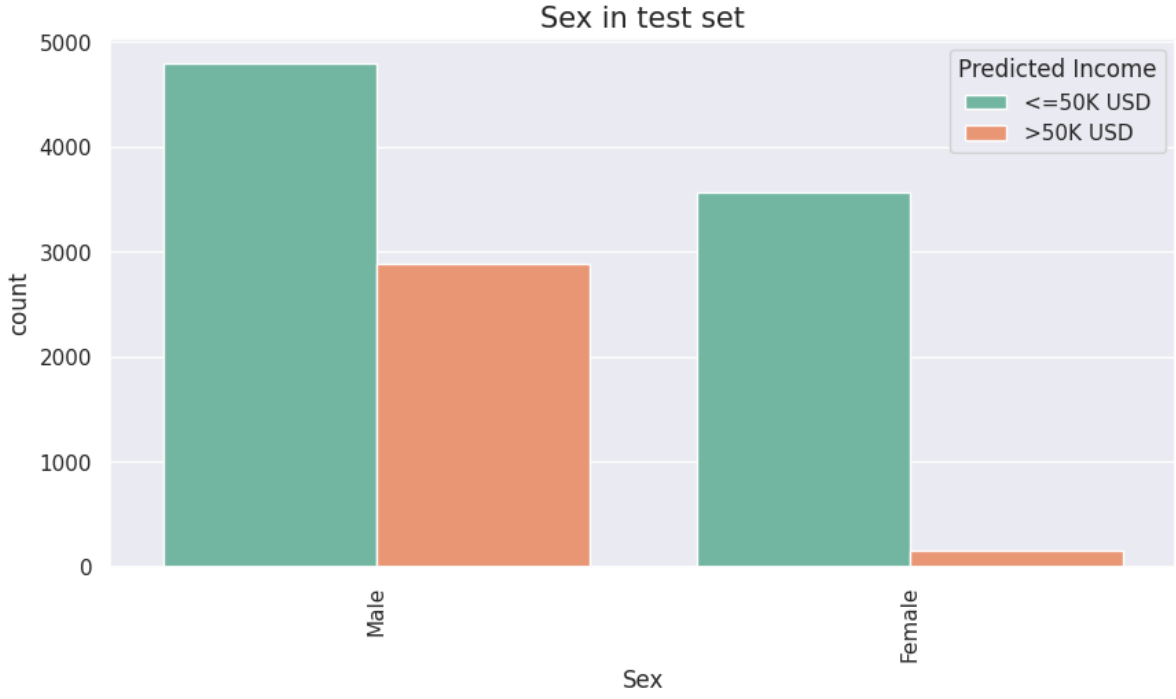

Education in test set
Predicted Income
<=50K USD
>50K USD
count
0
500
1000
1500
2000
2500
3000
Some-college
Bachelors
10th
HS-grad
11th
Masters
Assoc-voc
Assoc-acdm
1st-4th
Preschool
Doctorate
7th-8th
5th-6th
Prof-school
9th
12th
Education

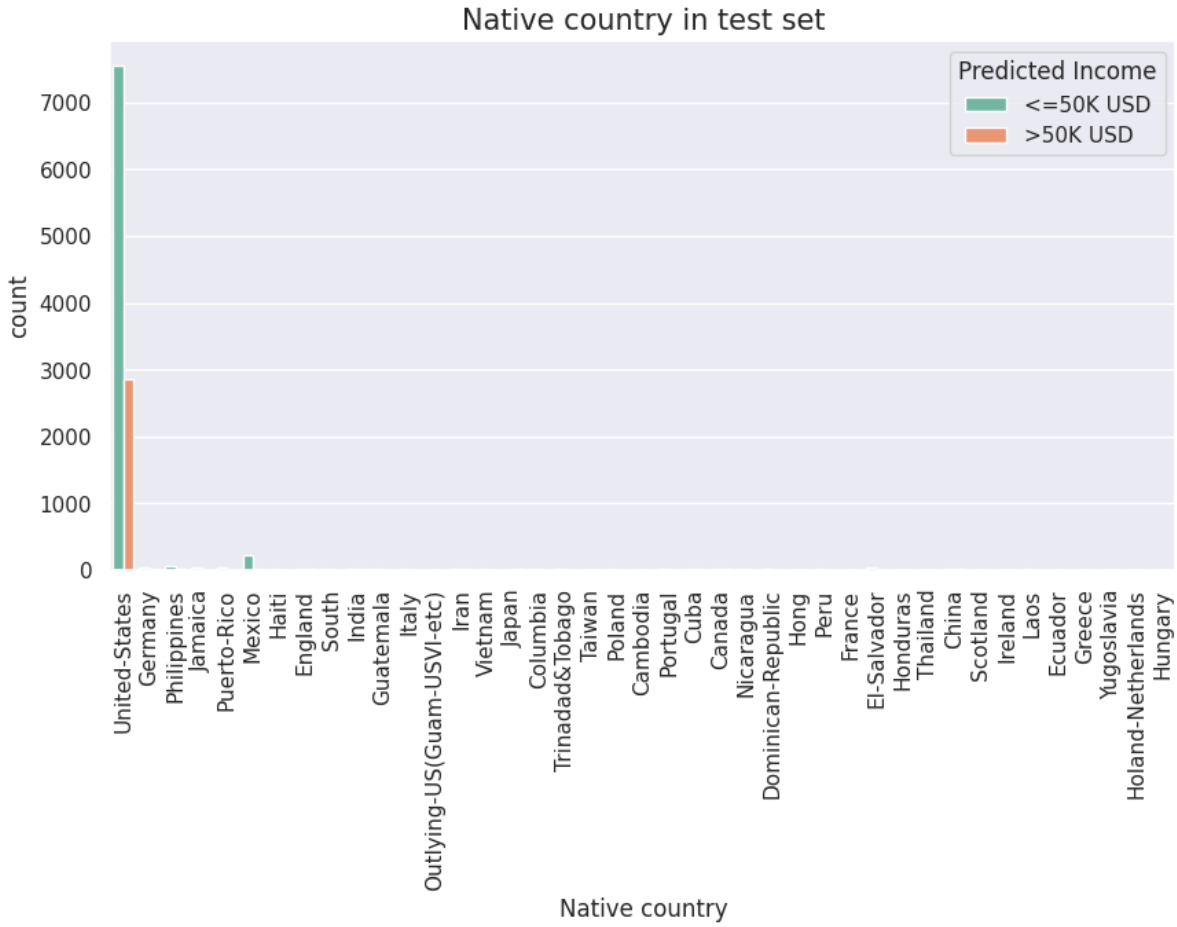

Marital status in test set
Predicted Income
<=50K USD
>50K USD
count
0
500
1000
1500
2000
2500
3000
3500
Married-civ-spouse
Widowed
Never-married
Divorced
Separated
Married-spouse-absent
Married-AF-spouse
Marital status


Occupation in test set
Predicted Income
<=50K USD
>50K USD
count
1400
1200
1000
800
600
400
200
0
Craft-repair
Farming-fishing
Exec-managerial
Other-service
Adm-clerical
Protective-serv
Sales
Machine-op-inspct
Transport-moving
Handlers-cleaners
Prof-specialty
Tech-support
Priv-house-serv
Armed-Forces
Occupation


Relationship in test set
Predicted Income
<=50K USD
>50K USD
count
2500
2000
1500
1000
500
0
Husband
Not-in-family
Own-child
Unmarried
Other-relative
Wife
Relationship


Race in test set
Predicted Income
<=50K USD
>50K USD
count
7000
6000
5000
4000
3000
2000
1000
0
White
Black
Asian-Pac-Islander
Amer-Indian-Eskimo
Other
Race


Sex in test set
Predicted Income
<=50K USD
>50K USD
count
5000
4000
3000
2000
1000
0
Male
Female
Sex



In [ ]:
```
#Checking the income for each country residents
for c in X_test['Native country'].unique():
  print(f"For {c}")
  print(X_test[X_test['Native country']==c]['Predicted Income'].value_counts())
```

```
For United-States
<=50K USD    7549
>50K USD     2849
Name: Predicted Income, dtype: int64
For Germany
<=50K USD    34
>50K USD     16
Name: Predicted Income, dtype: int64
For Philippines
<=50K USD    52
>50K USD     21
Name: Predicted Income, dtype: int64
For Jamaica
<=50K USD    31
>50K USD      1
Name: Predicted Income, dtype: int64
For Puerto-Rico
<=50K USD    41
>50K USD      1
Name: Predicted Income, dtype: int64
For Mexico
<=50K USD    225
>50K USD       8
Name: Predicted Income, dtype: int64
For Haiti
<=50K USD    15
>50K USD      1
Name: Predicted Income, dtype: int64
For England
<=50K USD    23
>50K USD     10
Name: Predicted Income, dtype: int64
For South
<=50K USD    22
>50K USD      6
Name: Predicted Income, dtype: int64
For India
>50K USD     24
<=50K USD    18
Name: Predicted Income, dtype: int64
For Guatemala
<=50K USD    26
>50K USD      1
Name: Predicted Income, dtype: int64
For Italy
<=50K USD    18
>50K USD      6
Name: Predicted Income, dtype: int64
For Outlying-US(Guam-USVI-etc)
<=50K USD    3
Name: Predicted Income, dtype: int64
For Iran
>50K USD     9
<=50K USD    8
Name: Predicted Income, dtype: int64
For Vietnam
<=50K USD    21
>50K USD      2
Name: Predicted Income, dtype: int64
For Japan
>50K USD     12
<=50K USD     7
Name: Predicted Income, dtype: int64
For Columbia
```

```
<=50K USD    15
>50K USD      2
Name: Predicted Income, dtype: int64
For Trinadad&Tobago
<=50K USD    9
>50K USD     1
Name: Predicted Income, dtype: int64
For Taiwan
<=50K USD    7
>50K USD     4
Name: Predicted Income, dtype: int64
For Poland
<=50K USD    19
>50K USD      6
Name: Predicted Income, dtype: int64
For Cambodia
<=50K USD    5
>50K USD     2
Name: Predicted Income, dtype: int64
For Portugal
<=50K USD    11
>50K USD      1
Name: Predicted Income, dtype: int64
For Cuba
<=50K USD    23
>50K USD      8
Name: Predicted Income, dtype: int64
For Canada
<=50K USD    28
>50K USD     20
Name: Predicted Income, dtype: int64
For Nicaragua
<=50K USD    13
>50K USD      1
Name: Predicted Income, dtype: int64
For Dominican-Republic
<=50K USD    21
Name: Predicted Income, dtype: int64
For Hong
<=50K USD    6
>50K USD     3
Name: Predicted Income, dtype: int64
For Peru
<=50K USD    8
Name: Predicted Income, dtype: int64
For France
>50K USD     5
<=50K USD    2
Name: Predicted Income, dtype: int64
For El-Salvador
<=50K USD    33
>50K USD      2
Name: Predicted Income, dtype: int64
For Honduras
<=50K USD    6
Name: Predicted Income, dtype: int64
For Thailand
<=50K USD    4
>50K USD     1
Name: Predicted Income, dtype: int64
For China
<=50K USD    13
>50K USD     11
Name: Predicted Income, dtype: int64
```

```
For Scotland
<=50K USD    3
>50K USD     1
Name: Predicted Income, dtype: int64
For Ireland
<=50K USD    6
>50K USD     2
Name: Predicted Income, dtype: int64
For Laos
<=50K USD    8
Name: Predicted Income, dtype: int64
For Ecuador
<=50K USD    6
>50K USD     2
Name: Predicted Income, dtype: int64
For Greece
<=50K USD    7
>50K USD     4
Name: Predicted Income, dtype: int64
For Yugoslavia
>50K USD     3
<=50K USD    1
Name: Predicted Income, dtype: int64
For Holand-Netherlands
<=50K USD    1
Name: Predicted Income, dtype: int64
For Hungary
<=50K USD    2
>50K USD     1
Name: Predicted Income, dtype: int64
```

End of Code

DAL Assignment Decision tree (revised)

Importing the libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import copy
import graphviz
from scipy.stats import chi2_contingency
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score,r
import seaborn as sns
```

Load the data

In [2]:
```
df=pd.read_excel(r"/content/drive/MyDrive/DAL dataset/Assignment 4/car_evaluation.x
```

In [3]:
```
df.head()
```

Out[3]:

	0	1	2	3	4	5	6
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

Set the column names

In [4]:
```
df=df.set_axis(['Buying price','Maintenance cost','Number of doors','Number of pers
```

In [5]:
```
df
```

Out[5]:

	Buying price	Maintenance cost	Number of doors	Number of persons	lug_boot	safety	decision
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc
...	...	...	...	...	...	...	...
1723	low	low	5more	more	med	med	good
1724	low	low	5more	more	med	high	vgood
1725	low	low	5more	more	big	low	unacc
1726	low	low	5more	more	big	med	good
1727	low	low	5more	more	big	high	vgood

1728 rows × 7 columns

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Buying price       1728 non-null   object
 1   Maintenance cost   1728 non-null   object
 2   Number of doors    1728 non-null   object
 3   Number of persons  1728 non-null   object
 4   lug_boot           1728 non-null   object
 5   safety             1728 non-null   object
 6   decision           1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

In [7]:
```
df=df.astype(str)
```

In [8]:
```
for cols in df.columns:
  print(df[cols].value_counts())
```

```
vhigh    432
high     432
med      432
low      432
Name: Buying price, dtype: int64
vhigh    432
high     432
med      432
low      432
Name: Maintenance cost, dtype: int64
2        432
3        432
4        432
5more    432
Name: Number of doors, dtype: int64
2       576
4       576
more    576
Name: Number of persons, dtype: int64
small    576
med      576
big      576
Name: lug_boot, dtype: int64
low     576
med     576
high    576
Name: safety, dtype: int64
unacc    1210
acc       384
good       69
vgood      65
Name: decision, dtype: int64
```

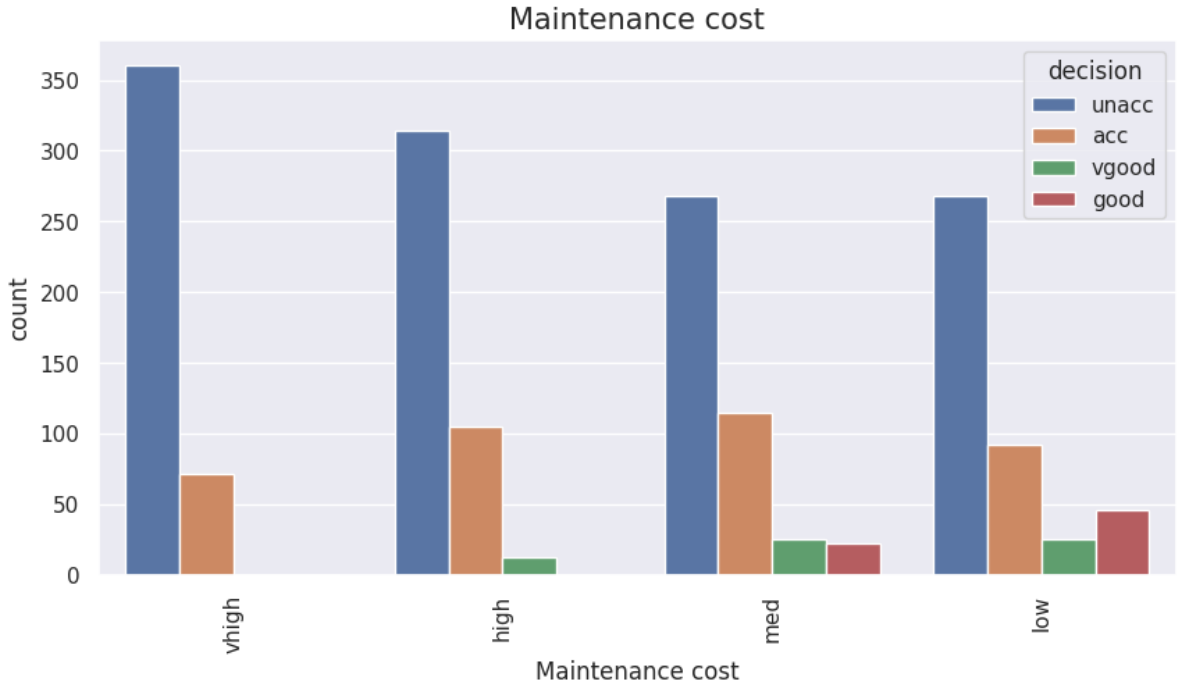
So there is no missing value in object type

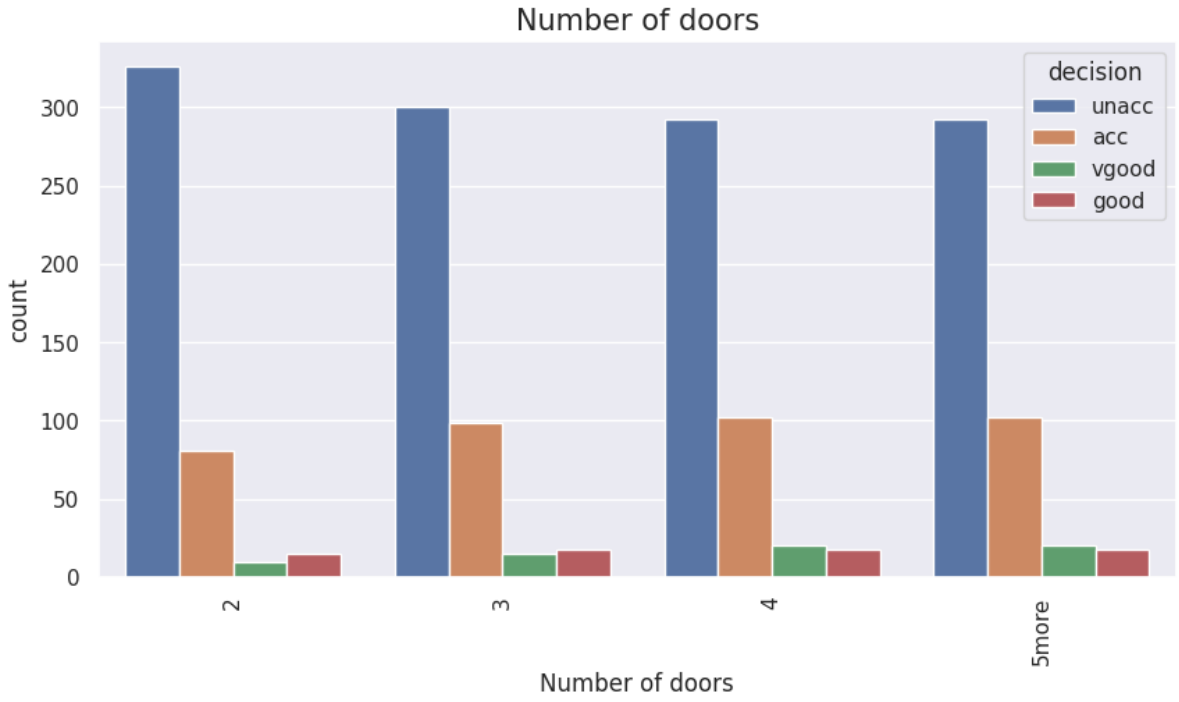
Preliminary visualization

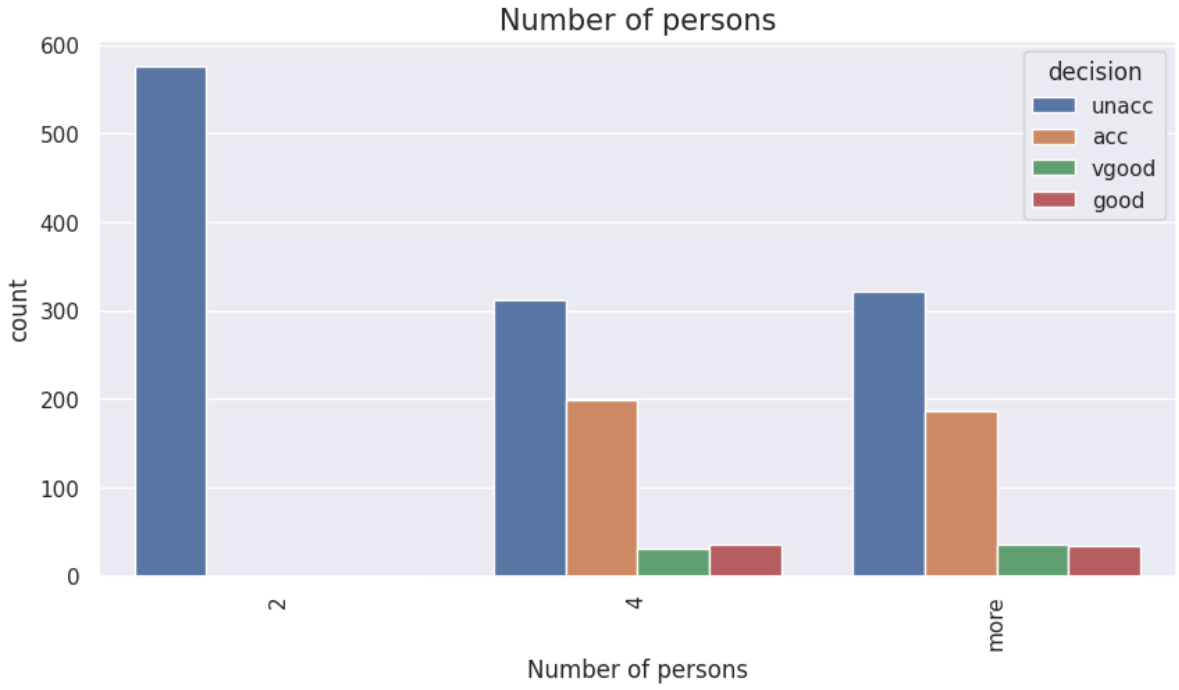
In [9]:

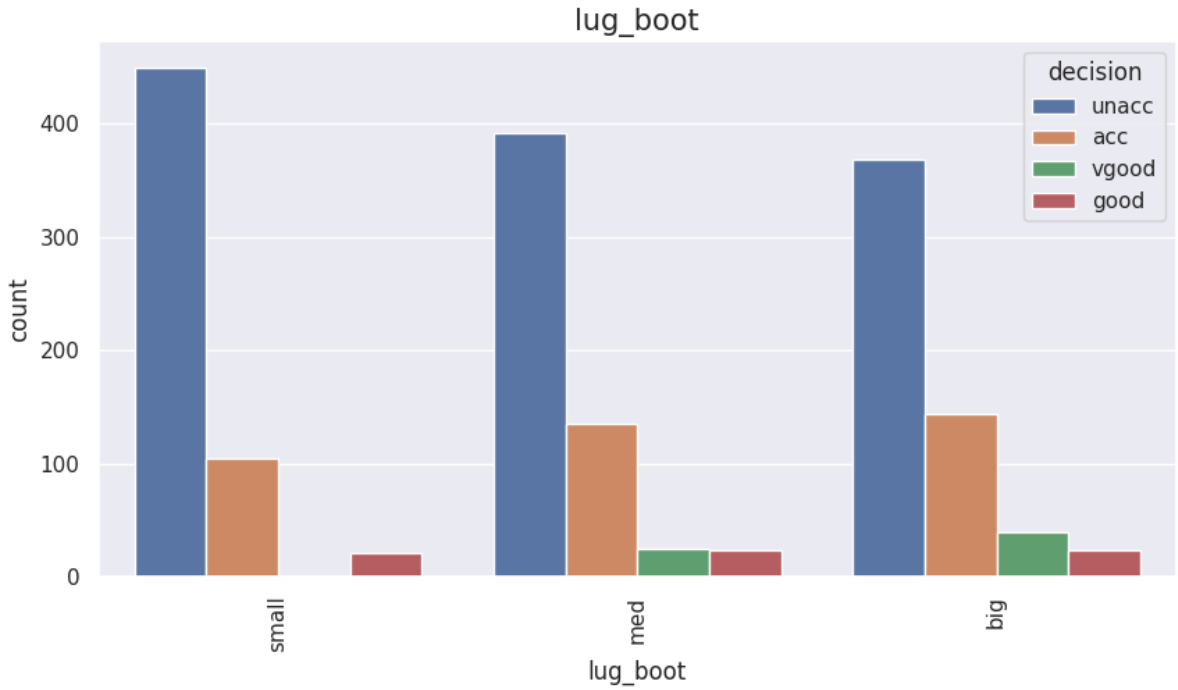
```
sns.set()
#check the bar plots for categorical features
for cat in df.columns:
  plt.figure(figsize=(10,5))
  sns.countplot(x=cat,data=df,hue='decision')
  plt.title(cat,fontsize=15)
  plt.xticks(rotation=90)
  plt.savefig(f"{cat}.png",bbox_inches = 'tight')
  plt.show()
```

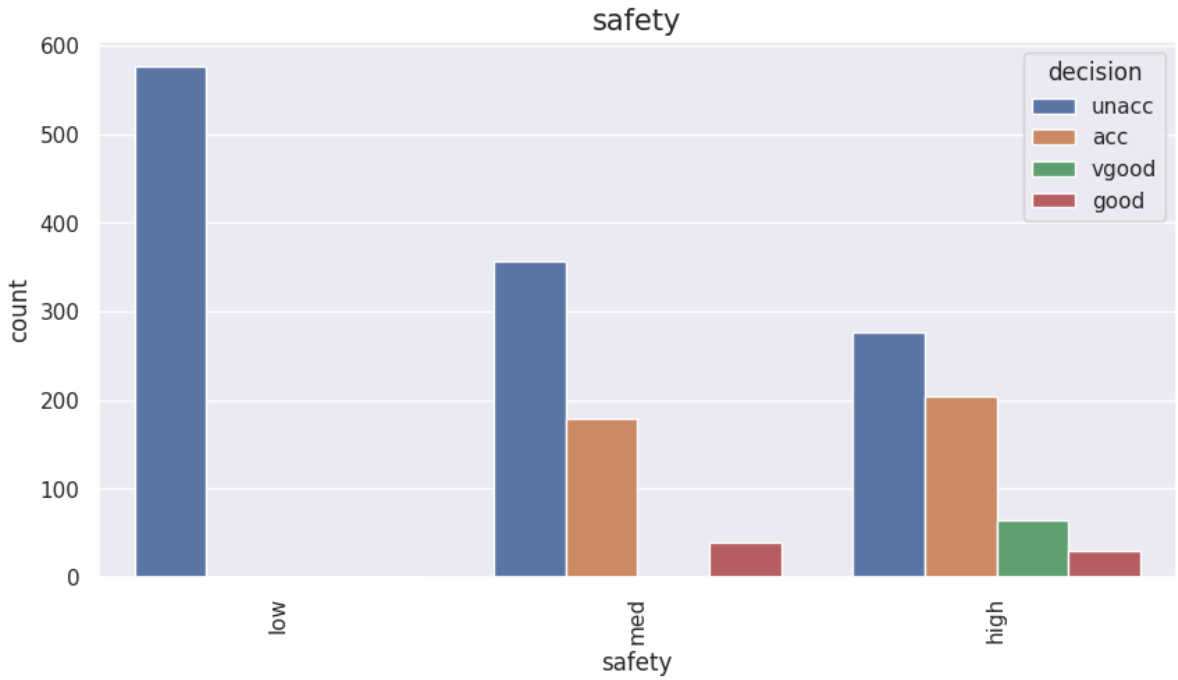

Buying price
decision
unacc
acc
vgood
good
count
0
50
100
150
200
250
300
350
vhigh
high
med
low
Buying price

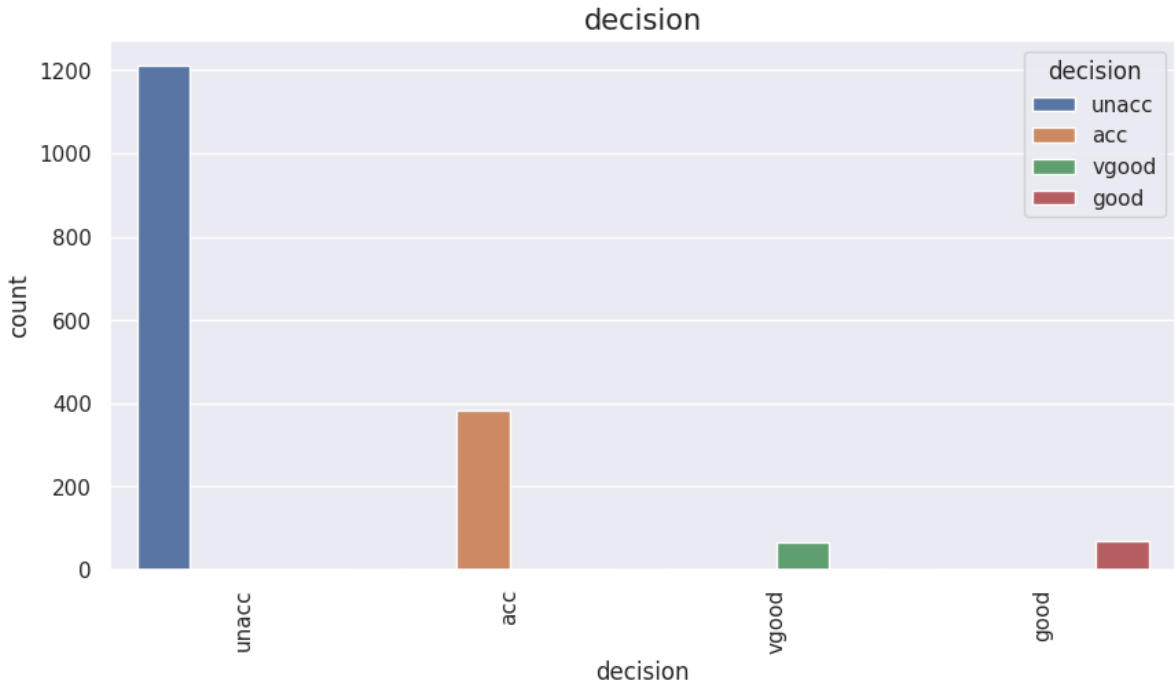

Maintenance cost
decision
unacc
acc
vgood
good
count
0
50
100
150
200
250
300
350
vhigh
high
med
low
Maintenance cost


Number of doors
decision
unacc
acc
vgood
good
count
0
50
100
150
200
250
300
2
3
4
5more
Number of doors


Number of persons
decision
unacc
acc
vgood
good
count
0
100
200
300
400
500
600
2
4
more
Number of persons


lug_boot
decision
unacc
acc
vgood
good
count
0
100
200
300
400
small
med
big
lug_boot

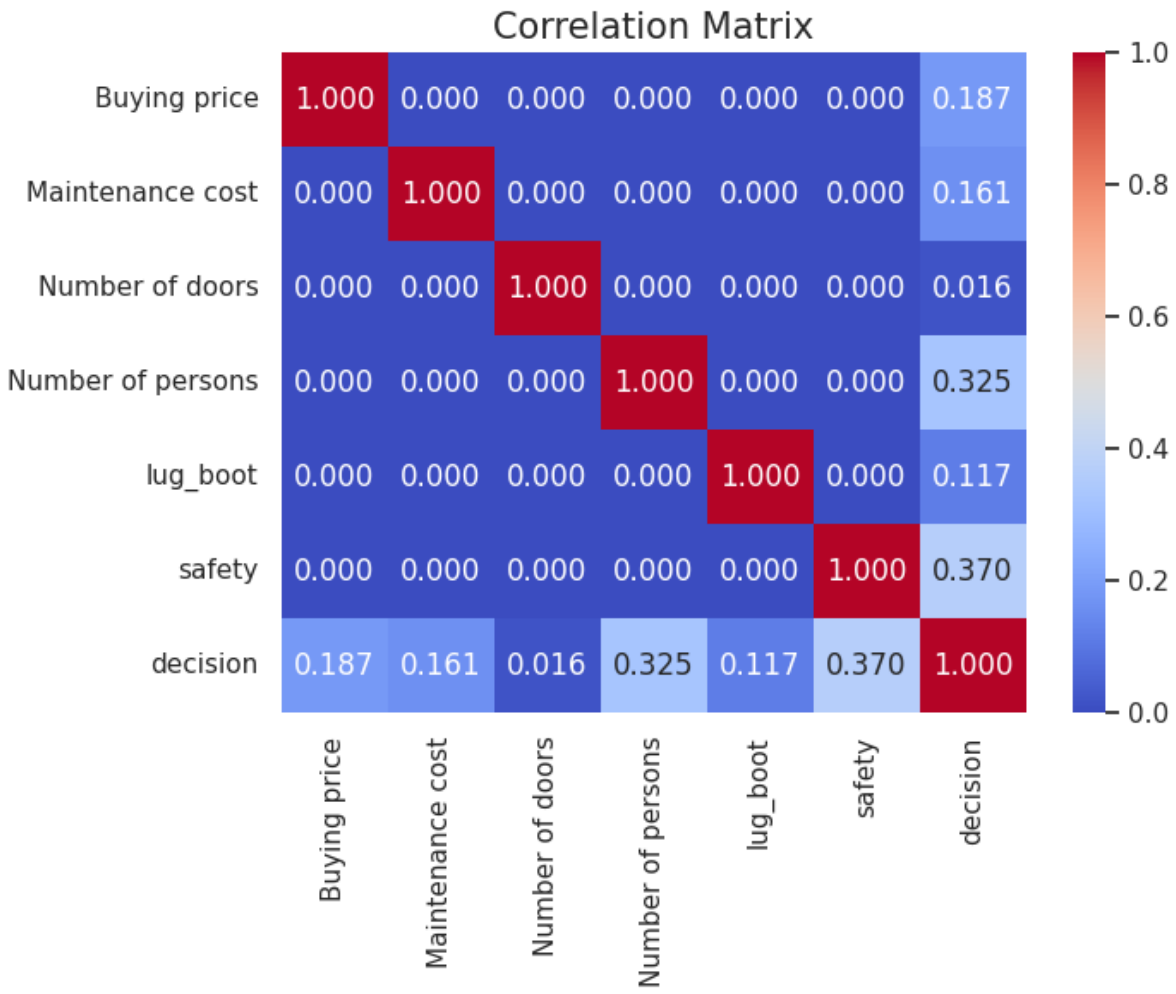

safety
decision
unacc
acc
vgood
good
count
0
100
200
300
400
500
600
low
med
high
safety



Cramer's V for correlation among categorical features

In [10]:

```
def cramers_v(confusion_matrix):
    chi2 = chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    phi2 = chi2 / n
    r, k = confusion_matrix.shape
    phi2corr = max(0, phi2 - ((k-1)*(r-1))/(n-1))
    rcorr = r - ((r-1)**2)/(n-1)
    kcorr = k - ((k-1)**2)/(n-1)
    return np.sqrt(phi2corr / min((kcorr-1), (rcorr-1)))

# Calculate correlation matrix for all features (including numerical and categorica
correlation_matrix = pd.DataFrame(index=df.columns, columns=df.columns, dtype=np.fl

for feature1 in df.columns:
    for feature2 in df.columns:
        if df[feature1].dtype == 'object' and df[feature2].dtype == 'object':
            # Calculate Cramér's V for two categorical variables
            conf = pd.crosstab(df[feature1], df[feature2])
            correlation_matrix.loc[feature1, feature2] = cramers_v(conf.values)

# Visualize the correlation matrix using Seaborn heatmap
plt.figure(figsize=(7,5))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".3f")
plt.title("Correlation Matrix",fontsize=15)
plt.show()
```



Independent features are almost uncorrelated. And the Decision feature is clearly dependent on the other features. So this data is really perfect to work. Without any necessity of feature selection

In [11]:
```
#drop the target
y=df.pop('decision')
X=df
```

In [12]:
```
#splitting the data
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42,st
```

Decision tree model class

In [13]:
```
#Define an utility class for decision tree
class DecisionTree:
  """
  Decision tree custom class
  """
  def _init_(tr):
    pass
  def label_enc(tr,A):
    """
    Encode the data
    Args : custom class object, input data
    """
    cp_A=copy.deepcopy(A)
    cp_A=cp_A.apply(LabelEncoder().fit_transform)
    return cp_A
```

```
  def fit(tr,A,b):
    """
    Fit the model
    Args : custom class object, input data, target
    """
    cp_A=copy.deepcopy(A)
    cp_A=tr.label_enc(cp_A)
    params={'max_depth':[7,10,13,16],'min_samples_split':[10,7,4,2],"max_leaf_nodes
    clf=DecisionTreeClassifier(criterion='gini')
    G=GridSearchCV(estimator=clf,param_grid=params,n_jobs=-1,cv=10)
    G.fit(cp_A,b)
    print(G.best_params_)
    tr.clf=G.best_estimator_
    return tr.clf
  def predict(tr,A):
    """
    Predicts the data
    Args : custom class object, input data
    """
    D=tr.clf.predict(tr.label_enc(A))
    D_p=tr.clf.predict_proba(tr.label_enc(A))
    return D,D_p
  def score(tr,b1,b2):
    """
    Predicts the score
    Args : custom class object, true label, predicted label
    """
    sc=accuracy_score(b1,b2)
    return sc
  def plot(tr,A,b):
    """
    Gets the tree plot
    Args : custom class object, input data, predicted label
    """
    cp_A=tr.label_enc(A)
    dot_data=tree.export_graphviz(tr.clf,out_file=None,feature_names=np.unique(cp_A
    graph = graphviz.Source(dot_data)
    return graph
  def cm(tr,y1,y2):
    """
    Plot the confusion matrix
    Args : custom class object, true label, predicted label
    """
    labels=np.unique(y1)
    CM=confusion_matrix(y1,y2)
    plt.figure(figsize=(10,8))
    plt.title("Confusion Matrix",fontsize=22)
    sns.heatmap(CM,annot=True,cmap='Pastel2',fmt='0.3g',linewidths=1.0)
    plt.yticks(np.arange(len(labels))+0.5,labels,fontsize=14,rotation='horizontal')
    plt.xticks(np.arange(len(labels))+0.5,labels,fontsize=14)
    plt.xlabel("Predicted",fontsize=20)
    plt.ylabel("Actual",fontsize=20)
    plt.savefig("Conf.png")
    plt.show()
```

In [14]:

```
#call the class and fit the model
E=DecisionTree()
E.fit(X_train,y_train)
```

```
{'max_depth': 13, 'max_leaf_nodes': 40, 'min_samples_split': 2}
```

Out[14]:
```
DecisionTreeClassifier(max_depth=13, max_leaf_nodes=40)
```

In [15]:
```
yhat_train,yhat_train_p=E.predict(X_train)[0],E.predict(X_train)[1]
```

In [16]:
```
E.score(y_train.values,yhat_train)#Get the accuracy score for train set
```

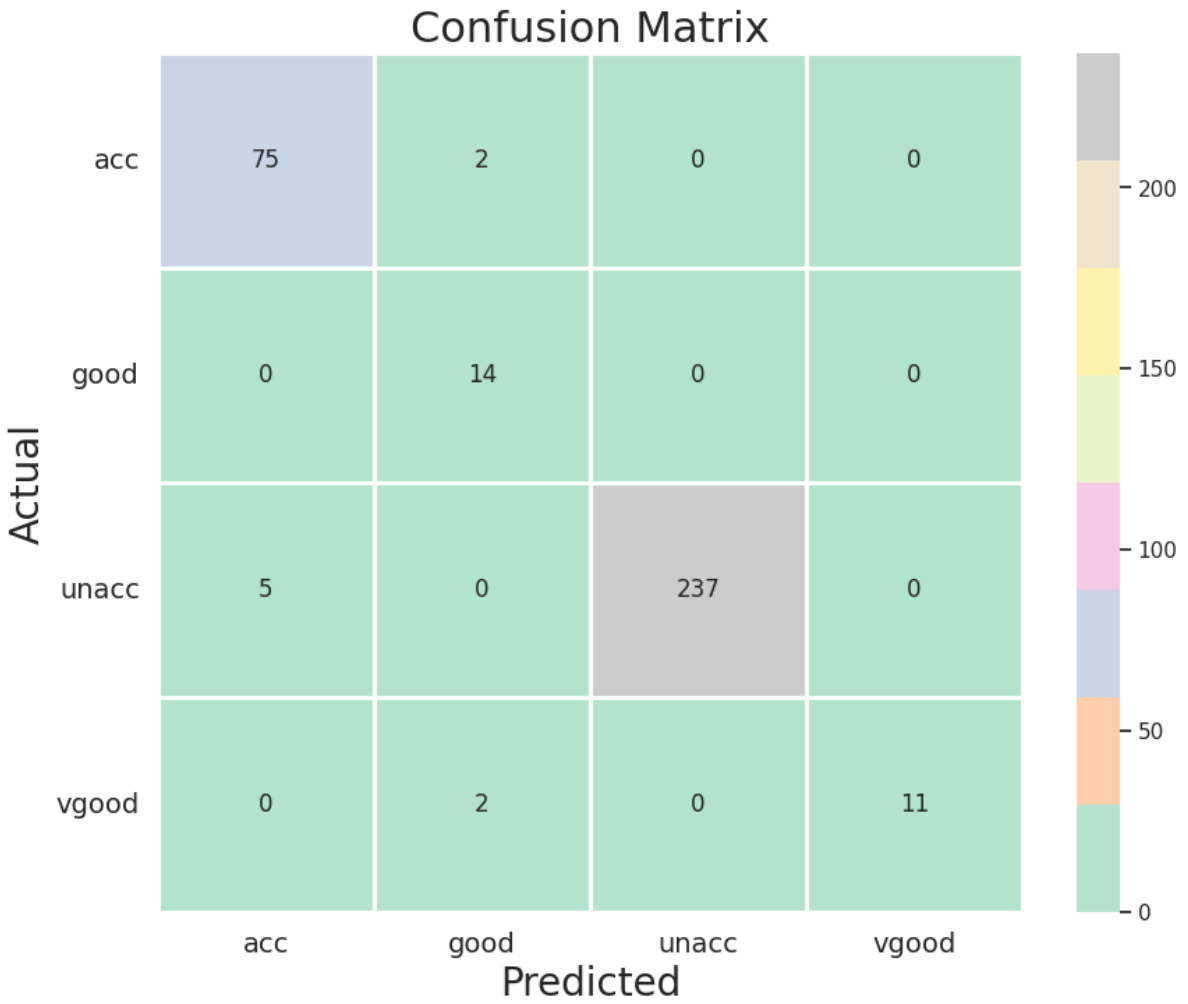
Out[16]:
```
0.9725036179450073
```

In [17]:
```
yhat_test,yhat_test_p=E.predict(X_test)[0],E.predict(X_test)[1]
E.score(y_test.values,yhat_test)#Get the accuracy for test set
```

Out[17]:
```
0.9739884393063584
```

In [18]:
```
#E.plot(X_train,y_train)
```

In [19]:
```
E.cm(y_test.values,yhat_test)#Check the confusion matrix
```



In [20]:
```
print(classification_report(y_test.values,yhat_test))#Classification metrics for th
```

```
              precision    recall  f1-score   support

         acc       0.94      0.97      0.96        77
        good       0.78      1.00      0.88        14
       unacc       1.00      0.98      0.99       242
       vgood       1.00      0.85      0.92        13

    accuracy                           0.97       346
   macro avg       0.93      0.95      0.93       346
weighted avg       0.98      0.97      0.97       346
```

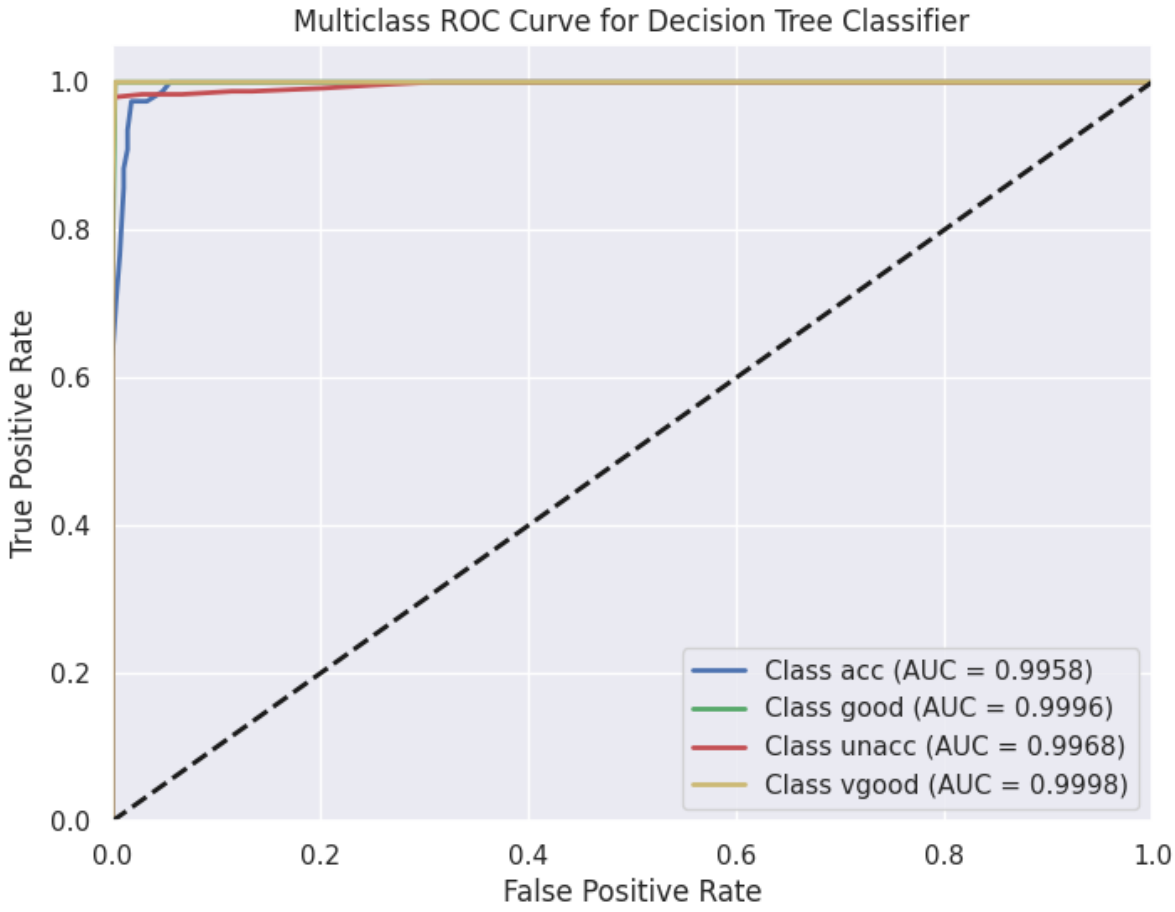
Get the one hot encoded data for ROC curve

In [21]:
```
oh_test=pd.get_dummies(y_test)
```

In [22]:
```
oh_pred=yhat_test_p
```

ROC curve

In [23]:
```
fpr = dict()
tpr = dict()
roc_auc = dict()
num_classes=oh_test.columns
for i in range(len(num_classes)):
    fpr[i], tpr[i], _ = roc_curve(oh_test.loc[:, num_classes[i]], oh_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot the ROC curves for each class
plt.figure(figsize=(8, 6))
colors = ['b', 'g', 'r','y']  # Use appropriate colors based on the number of class
for k,(i, color) in enumerate(zip(num_classes, colors)):
    plt.plot(fpr[k], tpr[k], color=color, lw=2, label=f'Class {i} (AUC = {roc_auc[k

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass ROC Curve for Decision Tree Classifier')
plt.legend(loc='lower right')
plt.savefig("ROC.png")
plt.show()
```

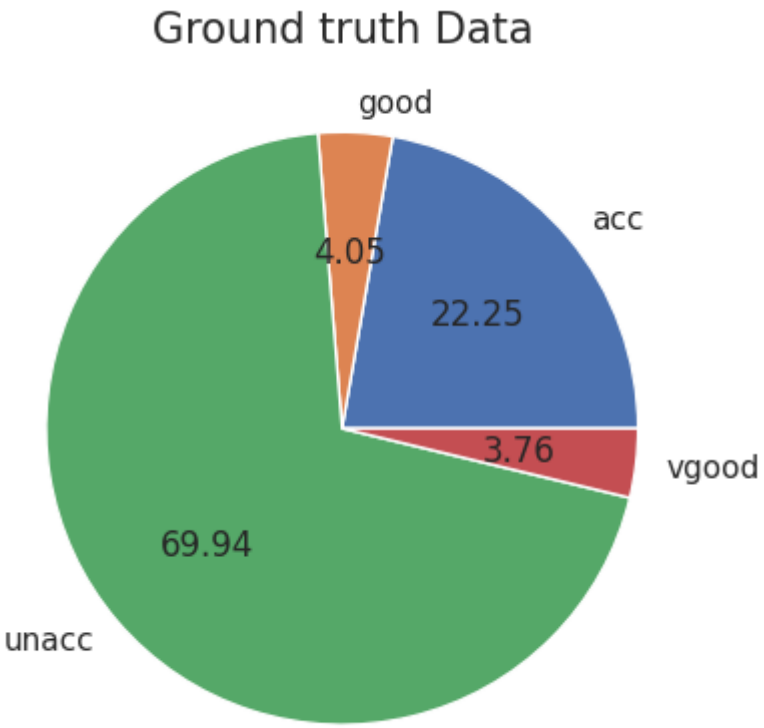


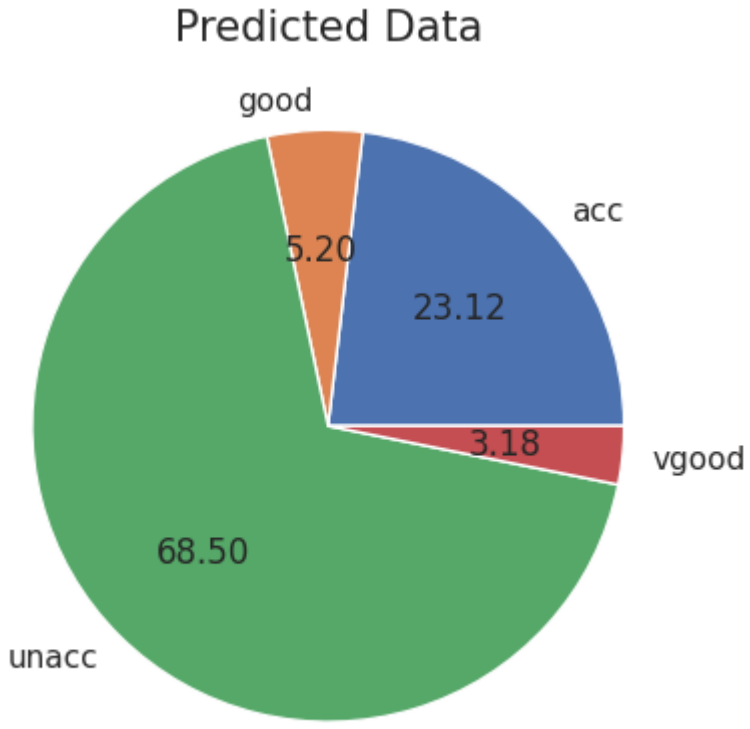
In [24]:
```
num_classes
```

Out[24]:
```
Index(['acc', 'good', 'unacc', 'vgood'], dtype='object')
```

Comparison of the plot in test data for labels between ground truth labels and predicted labels

In [25]:
```
Y_test=pd.DataFrame(y_test)
Y_pred=pd.DataFrame(yhat_test,columns=['decision'])
Y_test.groupby('decision').size().plot(kind='pie',autopct='%.2f')
plt.title("Ground truth Data",fontsize=15)
plt.show()
Y_pred.groupby('decision').size().plot(kind='pie', autopct='%.2f')
plt.title("Predicted Data",fontsize=15)
plt.show()
```



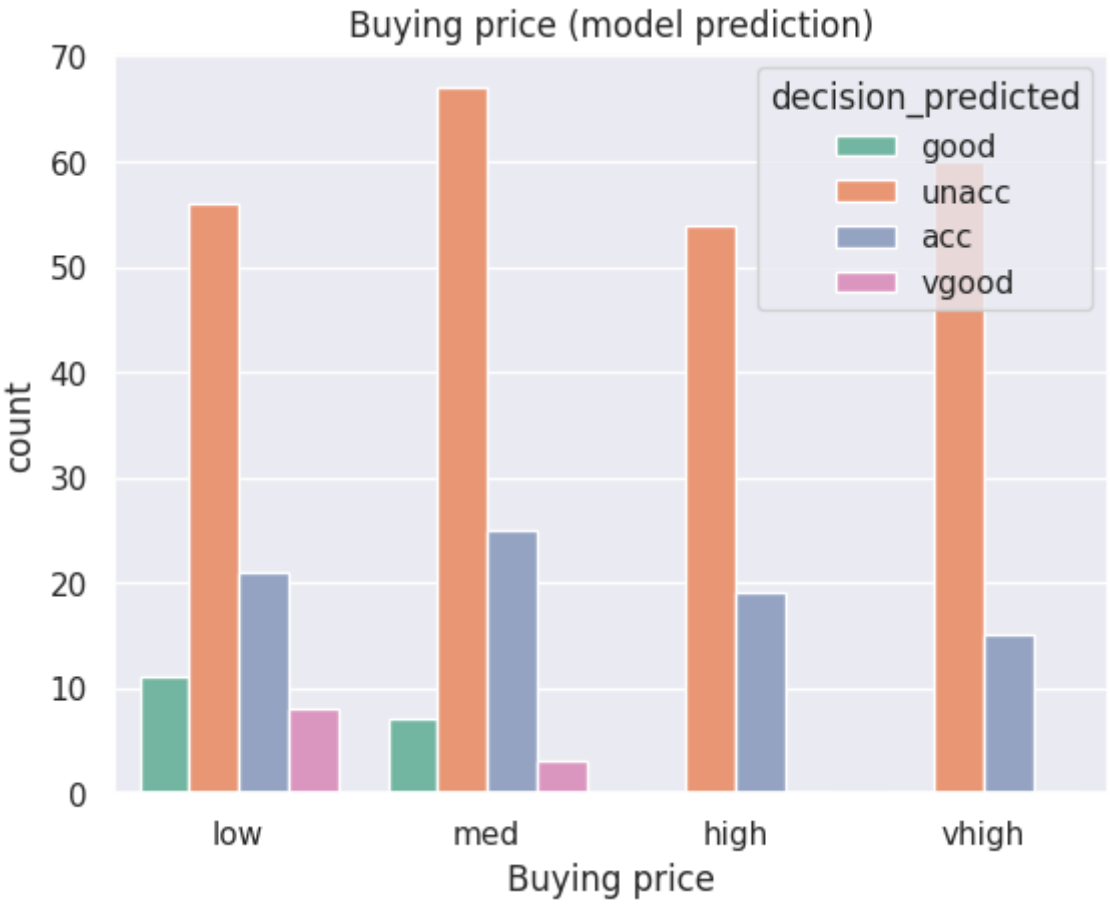


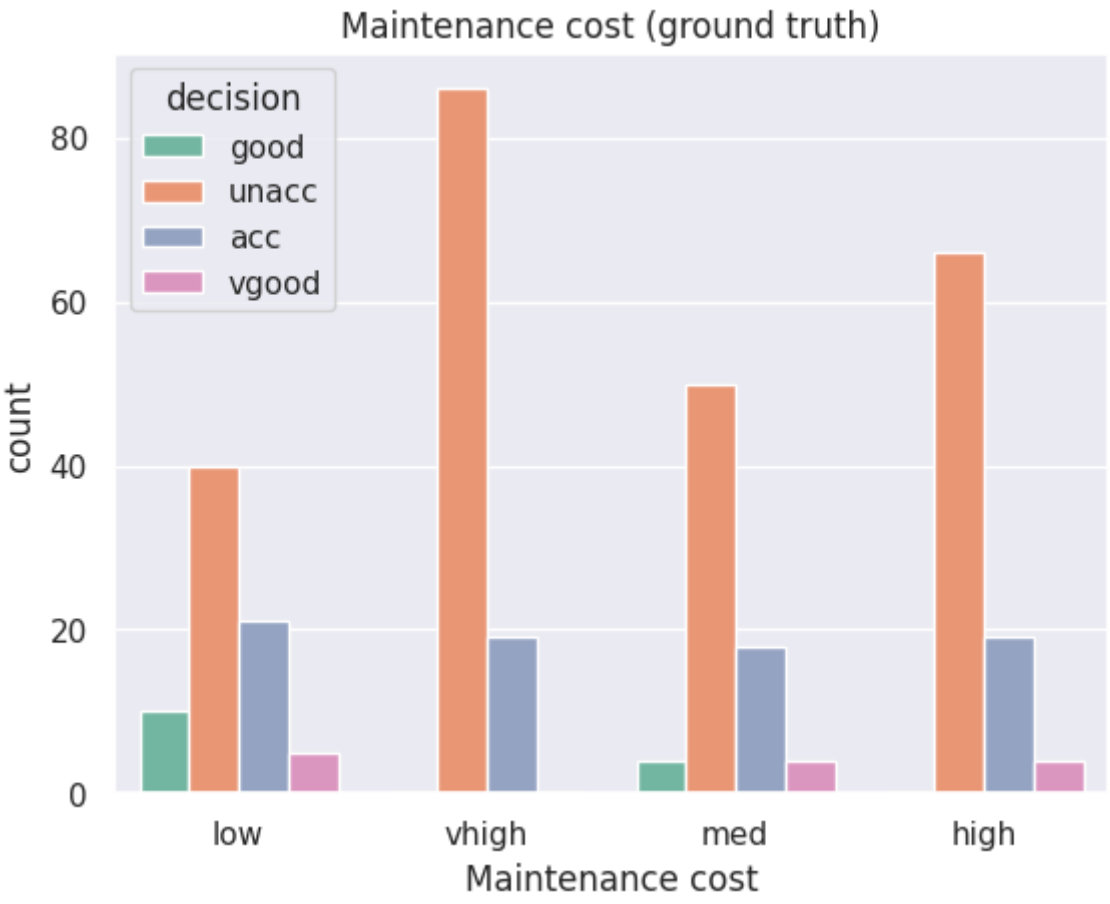
In [26]:
```
X_test1=copy.deepcopy(X_test)
X_test2=copy.deepcopy(X_test)
X_test1['decision']=y_test
X_test2['decision_predicted']=yhat_test
```

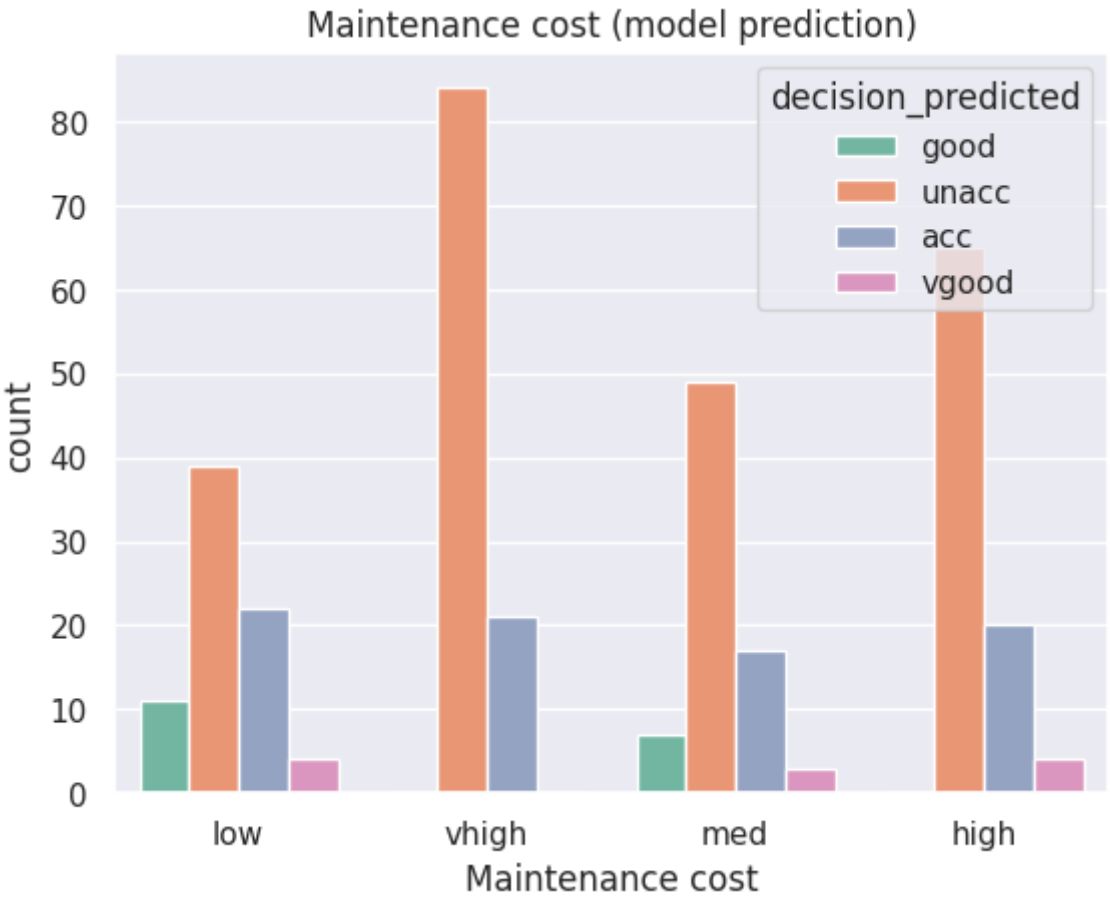
Verification of the test data

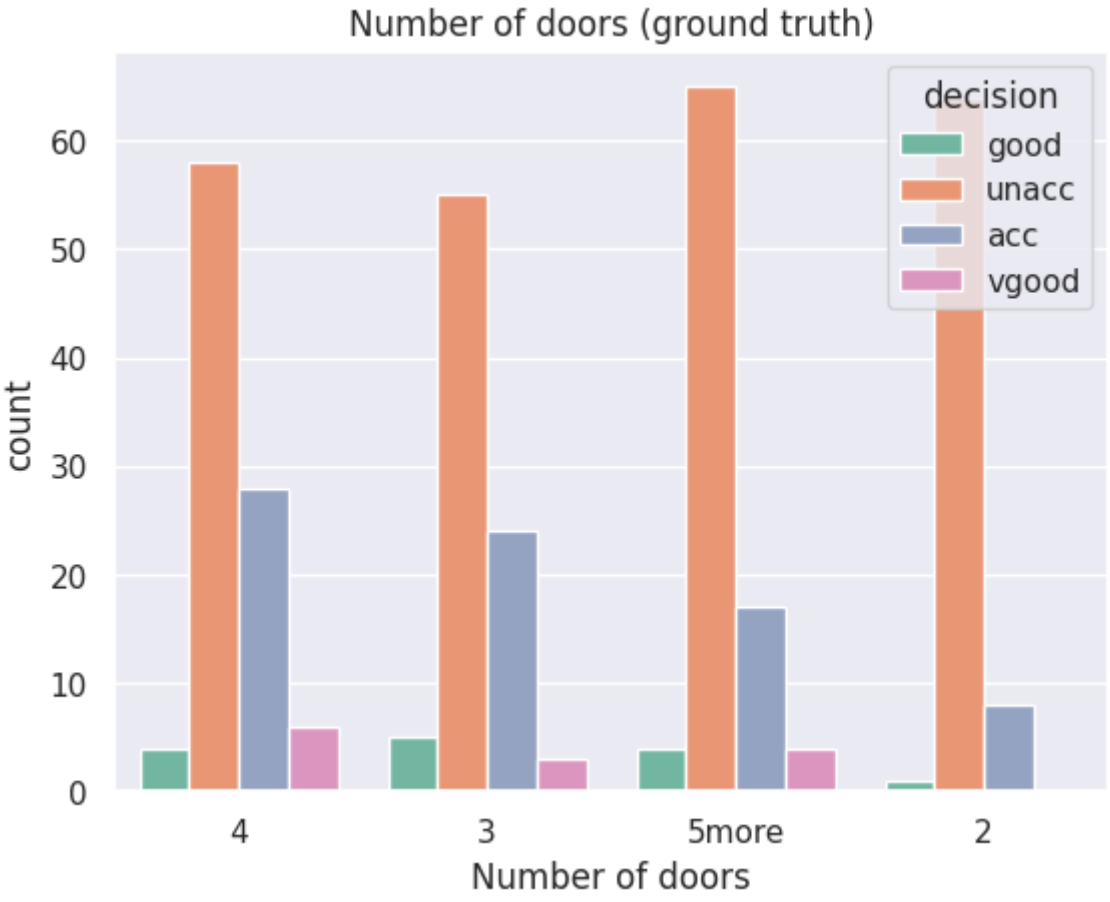
In [27]:
```
cols=['Buying price','Maintenance cost','Number of doors','Number of persons','lug_
for cat in cols:
  #fig, ax = plt.subplots(2,1 , figsize=(9, 10))
```

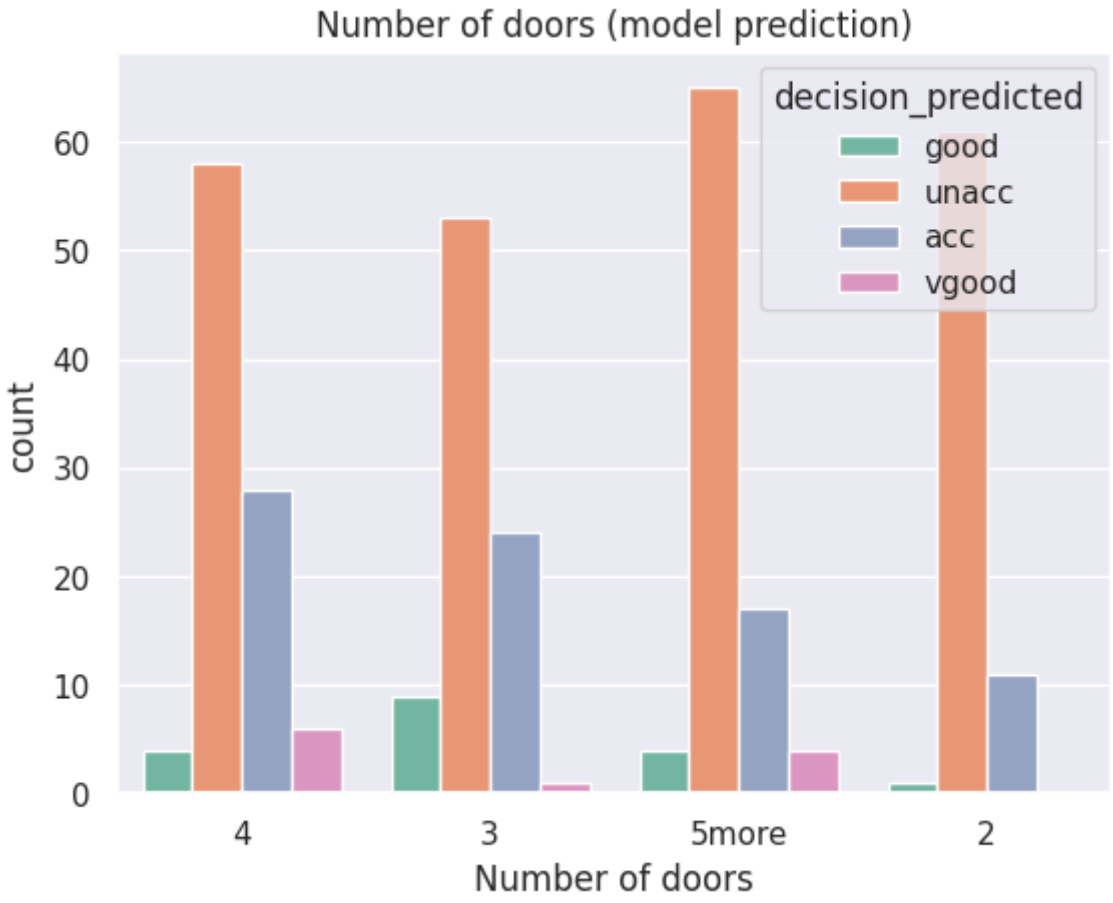
```
#fig.suptitle(cat,fontsize=20)
plt.title(f"{cat} (ground truth)")
sns.countplot(x=cat,data=X_test1,hue='decision',palette='Set2')
plt.savefig(f"test_{cat}_tru.png",bbox_inches = 'tight')
plt.show()
plt.title(f"{cat} (model prediction)")
sns.countplot(x=cat,data=X_test2,hue='decision_predicted',palette='Set2')
#ax[0].set_title("Ground truth case",fontsize=15)
#ax[1].set_title("Predicted case",fontsize=15)
#ax[0].set_xticklabels(labels=X_test1[cat].unique(),rotation=90)
#ax[1].set_xticklabels(labels=X_test1[cat].unique(),rotation=90)
plt.savefig(f"test_{cat}_pred.png",bbox_inches = 'tight')
plt.show()
```

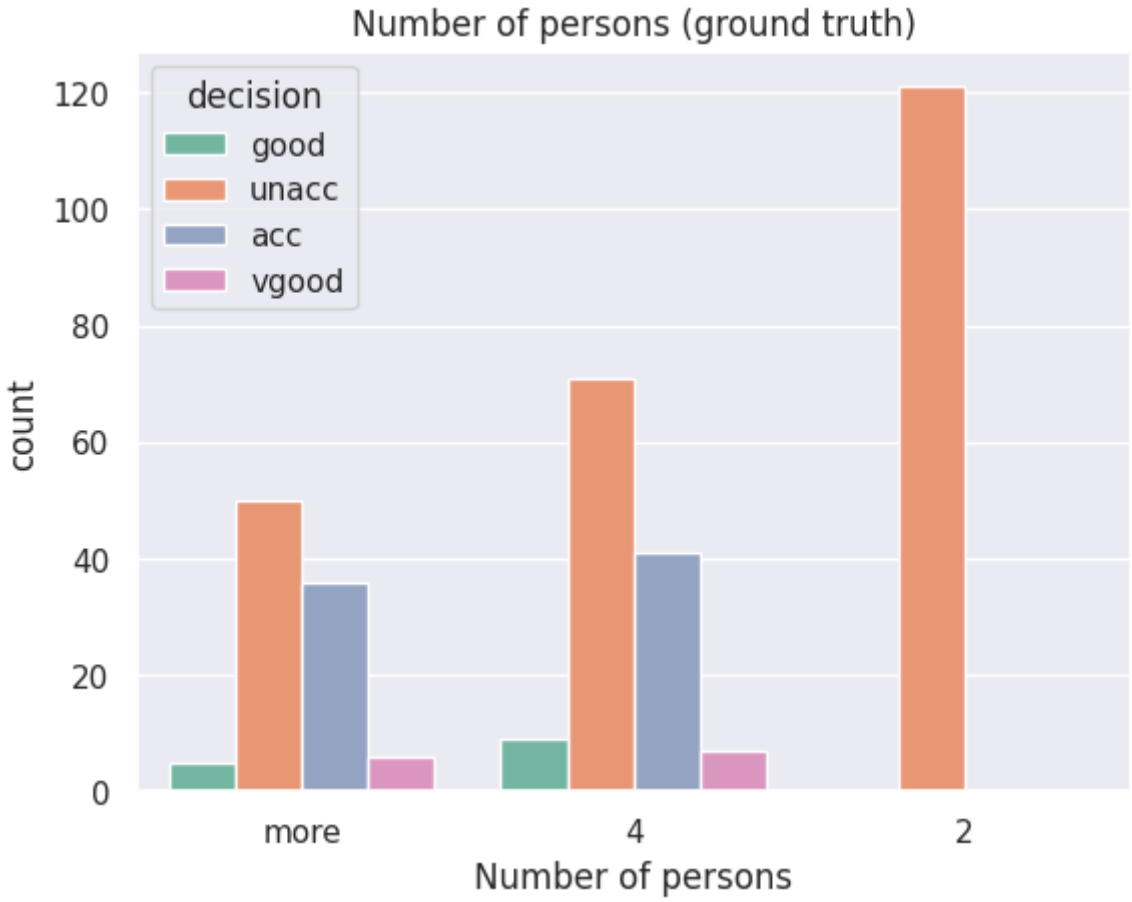

Buying price (model prediction)
decision_predicted
good
unacc
acc
vgood
count
0
10
20
30
40
50
60
70
low
med
high
vhigh
Buying price

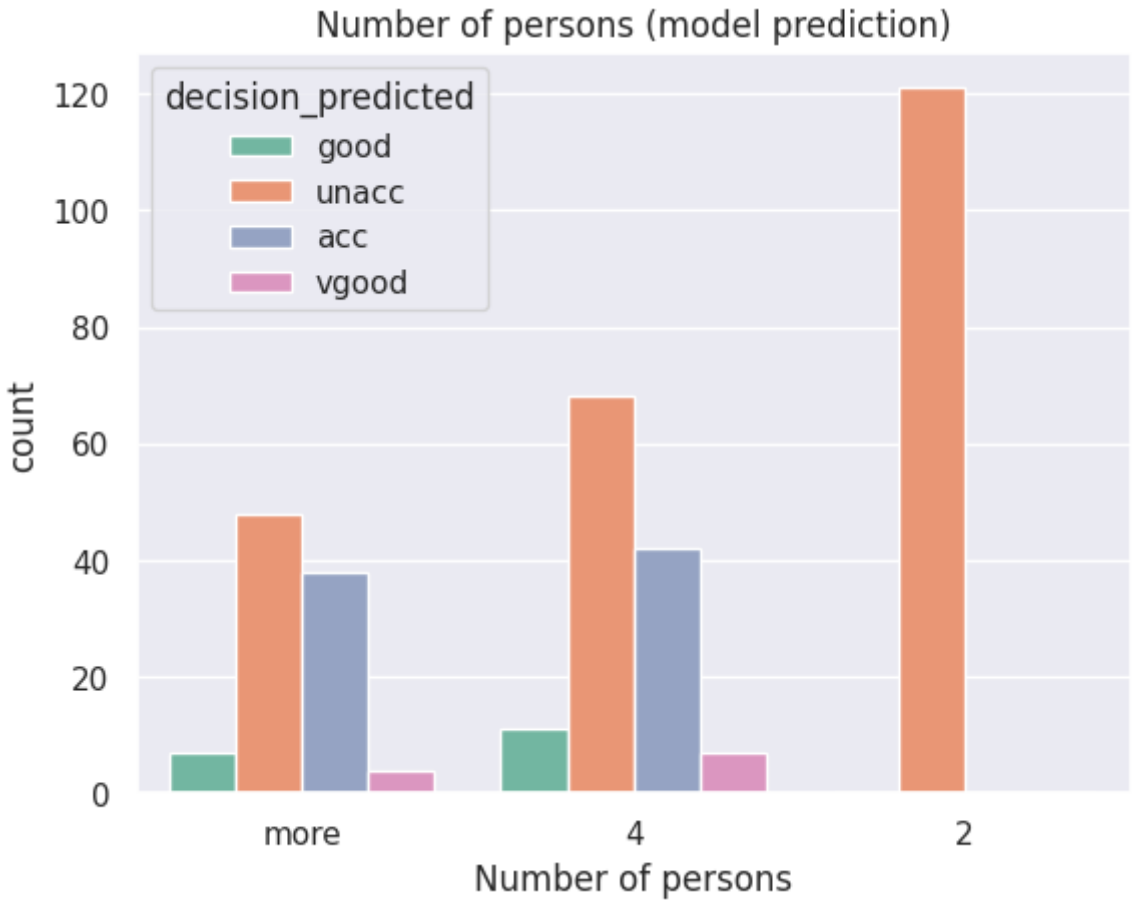

Maintenance cost (ground truth)
decision
good
unacc
acc
vgood
count
0
20
40
60
80
low
vhigh
med
high
Maintenance cost

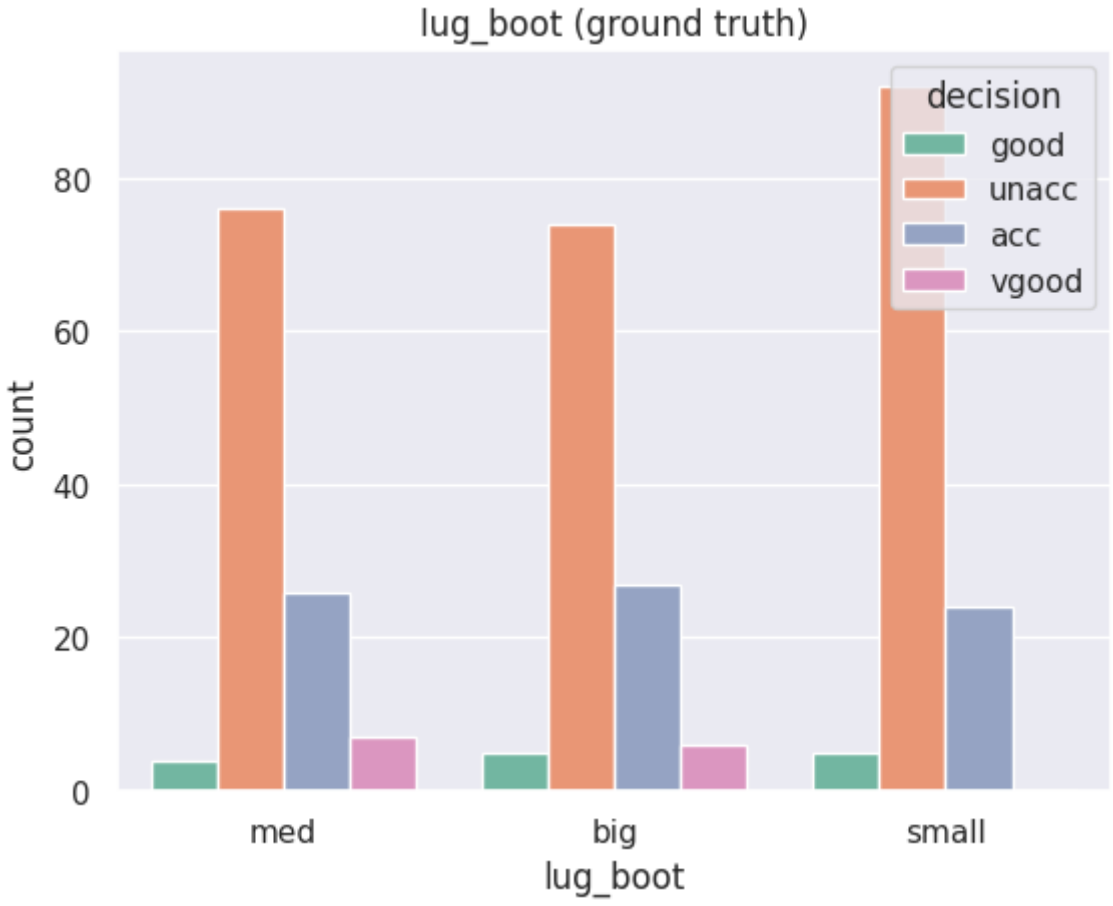

Maintenance cost (model prediction)
decision_predicted
good
unacc
acc
vgood
count
0
10
20
30
40
50
60
70
80
low
vhigh
med
high
Maintenance cost

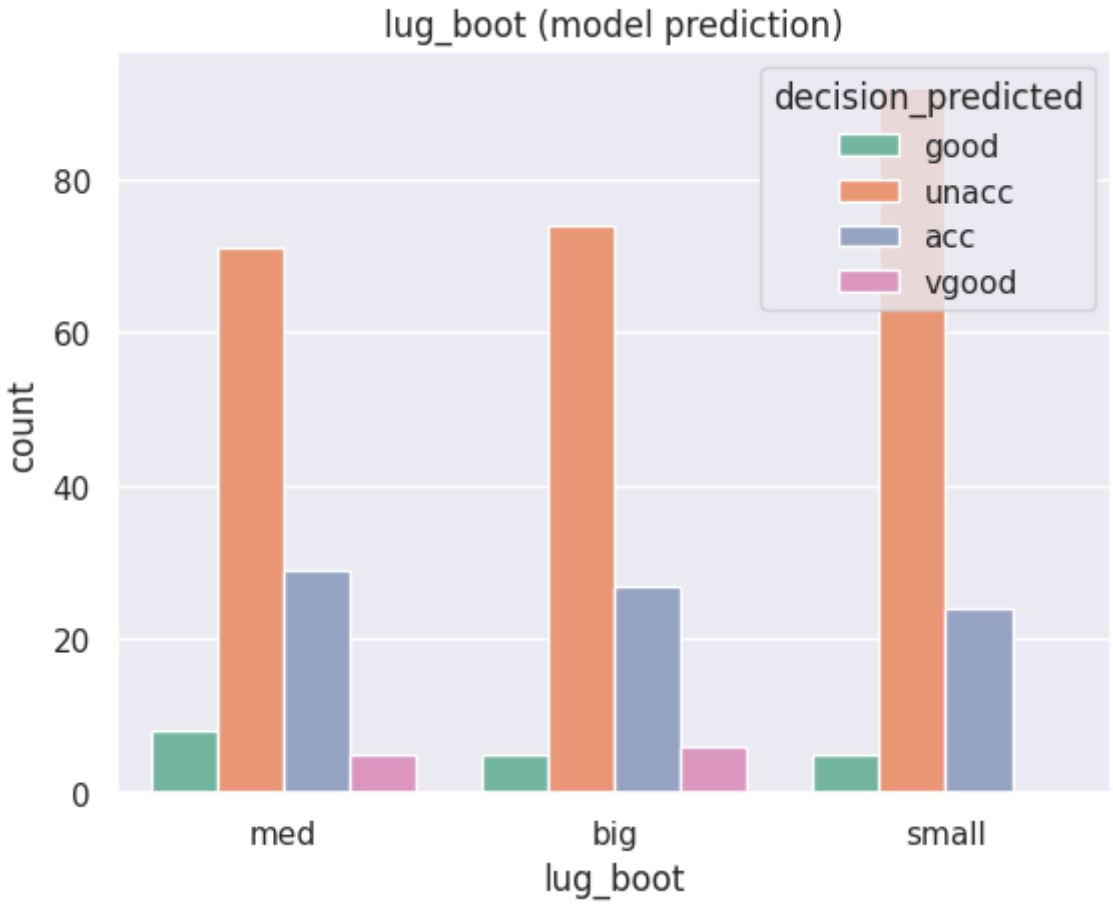

Number of doors (ground truth)
decision
good
unacc
acc
vgood
count
0
10
20
30
40
50
60
4
3
5more
2
Number of doors

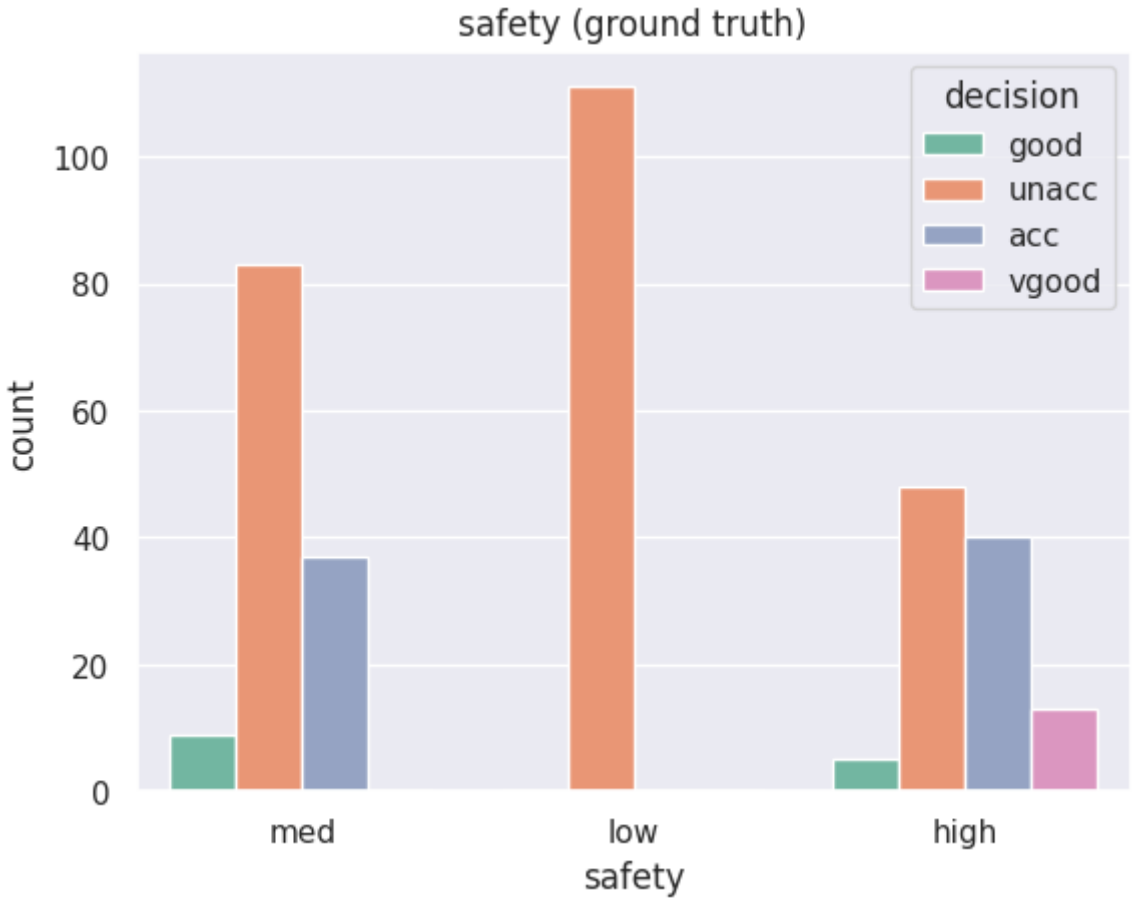

Number of doors (model prediction)
decision_predicted
good
unacc
acc
vgood
count
0
10
20
30
40
50
60
4
3
5more
2
Number of doors

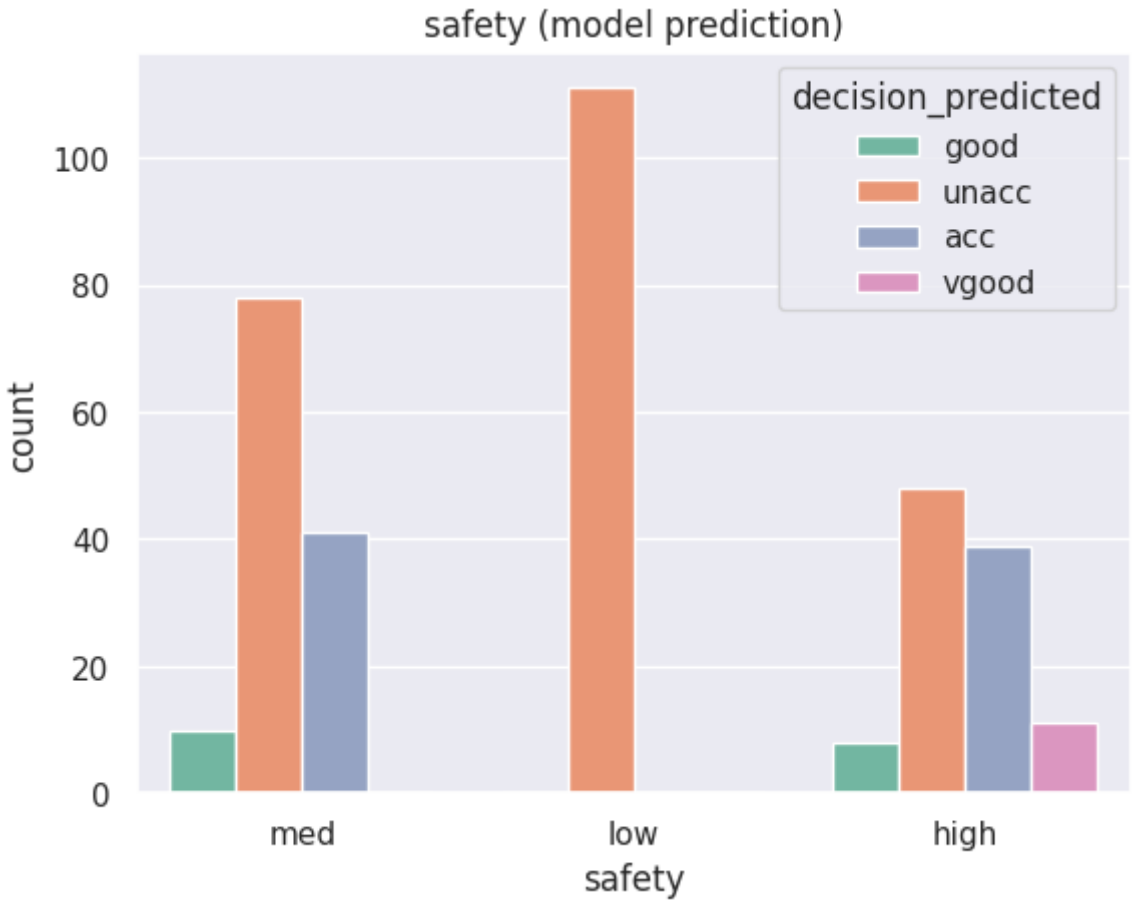

Number of persons (ground truth)
decision
good
unacc
acc
vgood
count
0
20
40
60
80
100
120
more
4
2
Number of persons


Number of persons (model prediction)
decision_predicted
good
unacc
acc
vgood
120
100
80
60
40
20
0
count
more
4
2
Number of persons


lug_boot (ground truth)
decision
good
unacc
acc
vgood
80
60
40
20
0
count
med
big
small
lug_boot


lug_boot (model prediction)
decision_predicted
good
unacc
acc
vgood
count
0
20
40
60
80
med
big
small
lug_boot


safety (ground truth)
decision
good
unacc
acc
vgood
count
0
20
40
60
80
100
med
low
high
safety


safety (model prediction)
decision_predicted
good
unacc
acc
vgood
100
80
60
40
20
0
count
med
low
high
safety

End of Code

DAL Assignment Random forest (revised)

Importing libraries

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import copy
import graphviz
from scipy.stats import chi2_contingency
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.compose import ColumnTransformer
from sklearn.base import TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn import tree
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score,r
import seaborn as sns
```

Read the data

In [ ]:
```
df=pd.read_excel(r"/content/drive/MyDrive/DAL dataset/Assignment 5/car_evaluation.x
df.head()
```

Out[ ]:

	0	1	2	3	4	5	6
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

In [ ]:
```
df=df.set_axis(['Buying price','Maintenance cost','Number of doors','Number of pers
```

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Buying price       1728 non-null   object
 1   Maintenance cost   1728 non-null   object
 2   Number of doors    1728 non-null   object
 3   Number of persons  1728 non-null   object
 4   lug_boot           1728 non-null   object
 5   safety             1728 non-null   object
 6   decision           1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

In [ ]:
```
df=df.astype(str)
for cols in df.columns:
  print(df[cols].value_counts())
```

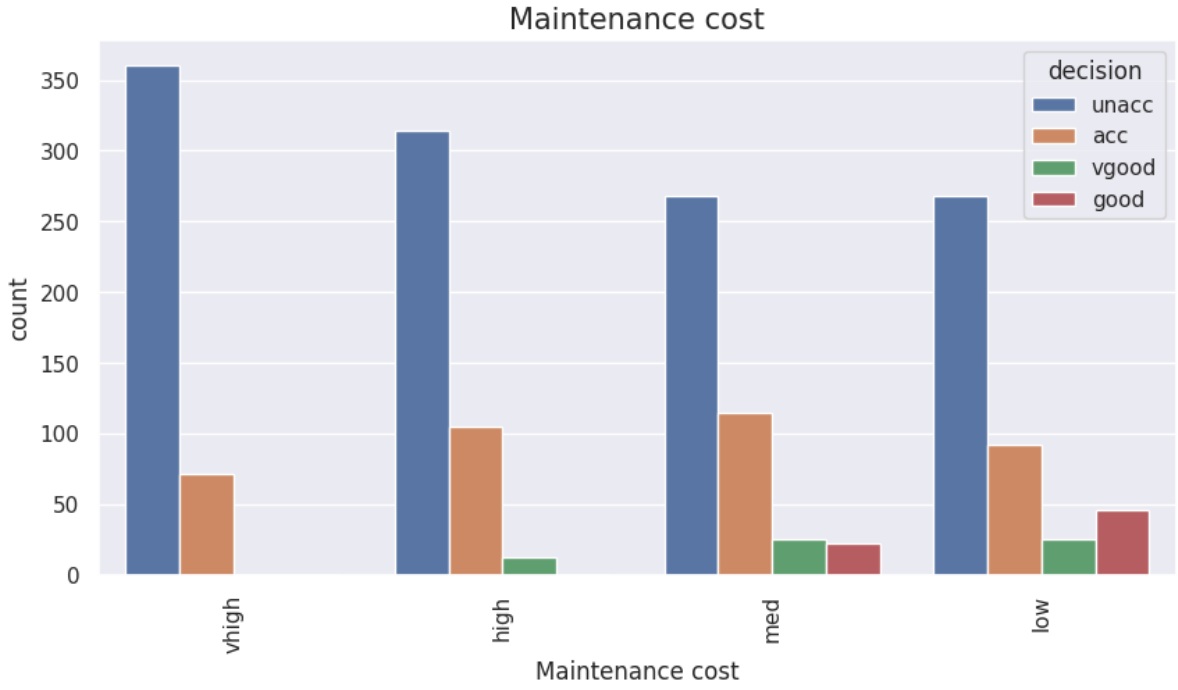
```
vhigh    432
high     432
med      432
low      432
Name: Buying price, dtype: int64
vhigh    432
high     432
med      432
low      432
Name: Maintenance cost, dtype: int64
2        432
3        432
4        432
5more    432
Name: Number of doors, dtype: int64
2       576
4       576
more    576
Name: Number of persons, dtype: int64
small    576
med      576
big      576
Name: lug_boot, dtype: int64
low     576
med     576
high    576
Name: safety, dtype: int64
unacc    1210
acc       384
good       69
vgood      65
Name: decision, dtype: int64
```

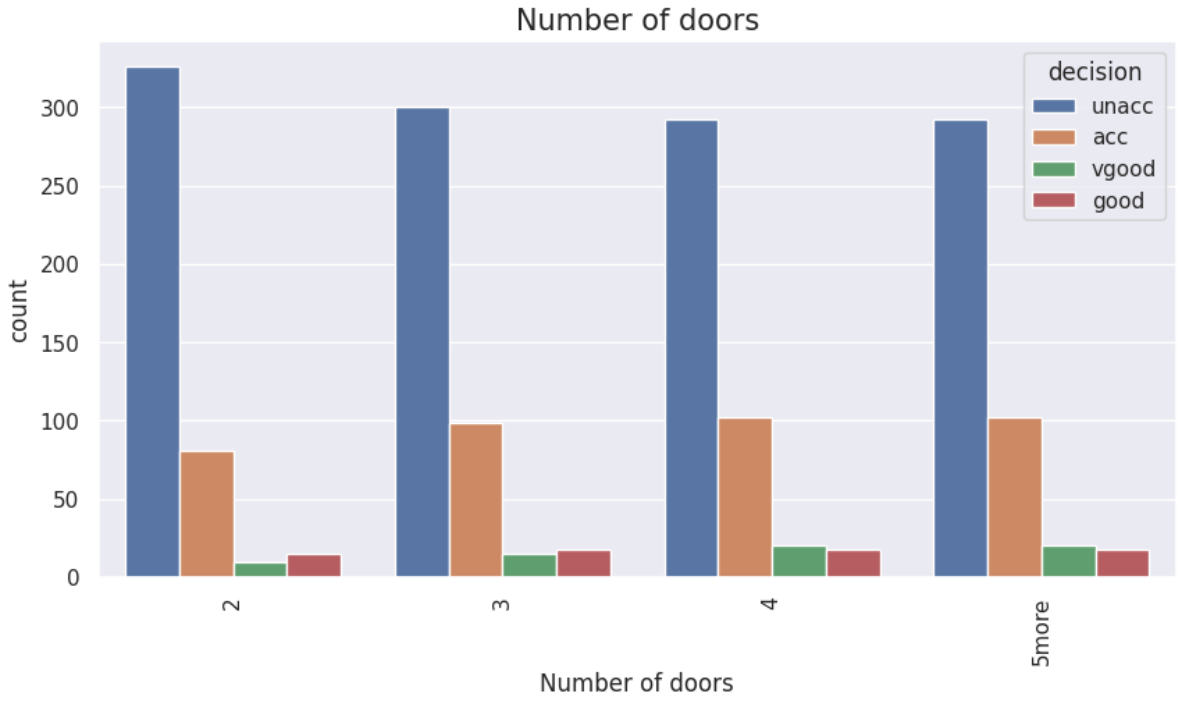
So there is no missing value in object type

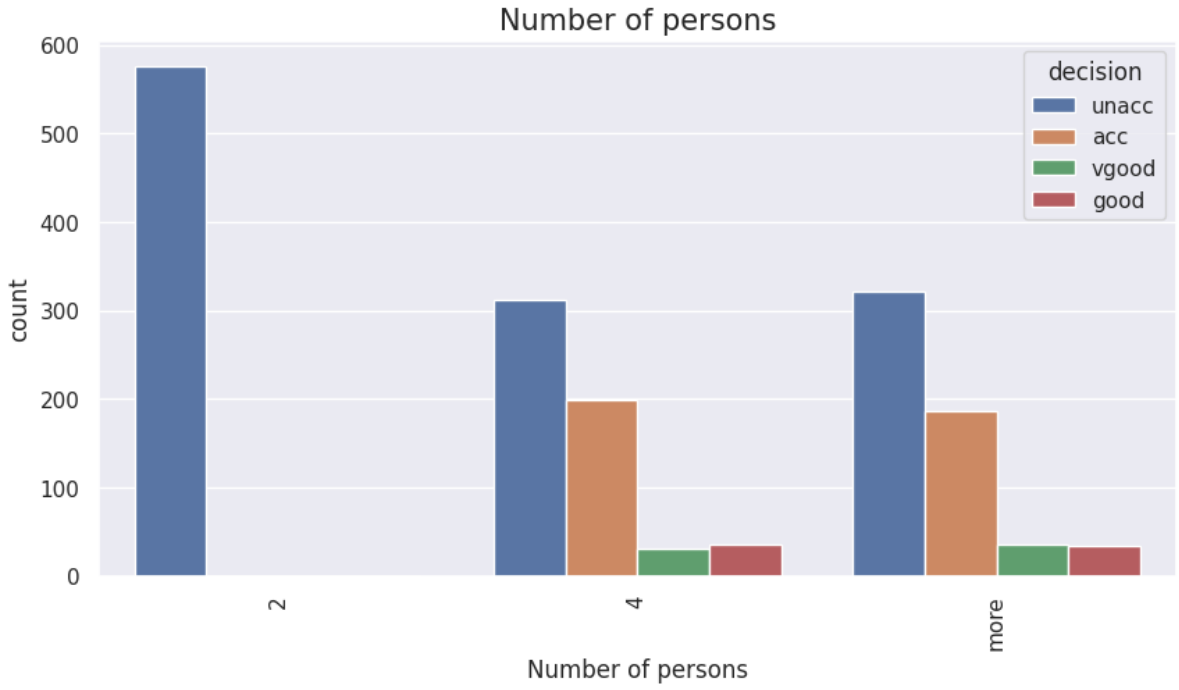
Preliminary visualization

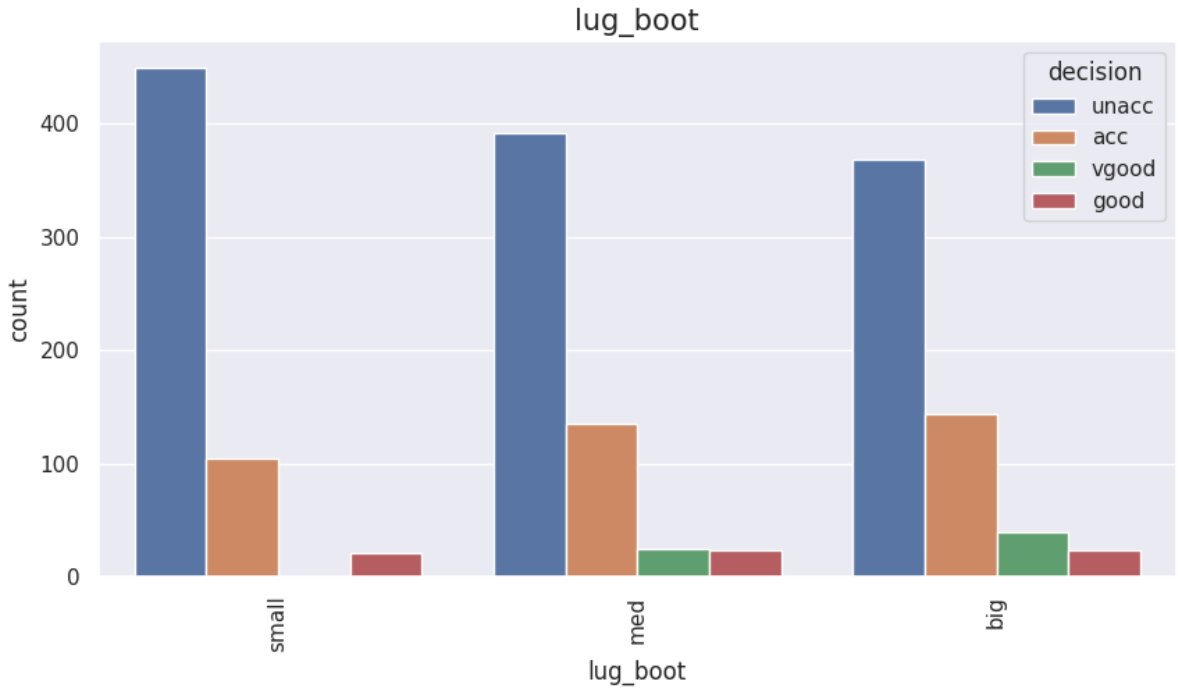
In [ ]:
```
sns.set()
#check the bar plots for categorical features
for cat in df.columns:
  plt.figure(figsize=(10,5))
  sns.countplot(x=cat,data=df,hue='decision')
  plt.title(cat,fontsize=15)
  plt.xticks(rotation=90)
  plt.savefig(f"{cat}.png",bbox_inches = 'tight')
  plt.show()
```

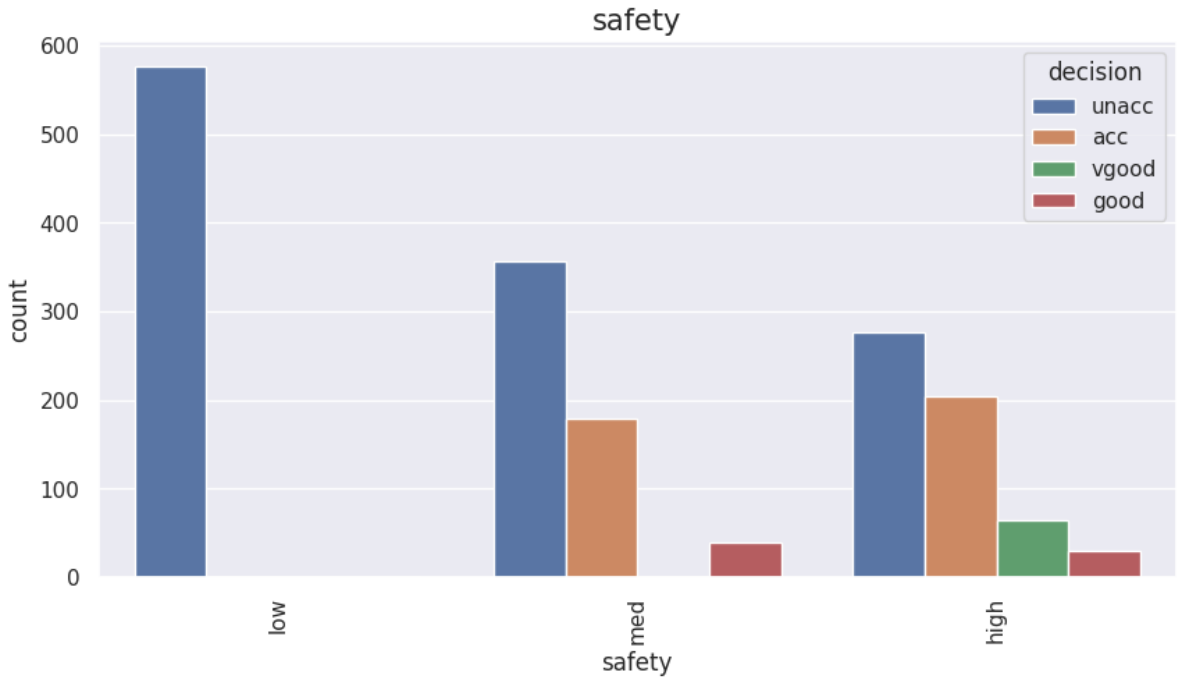

Buying price
decision
unacc
acc
vgood
good
count
0
50
100
150
200
250
300
350
vhigh
high
med
low
Buying price

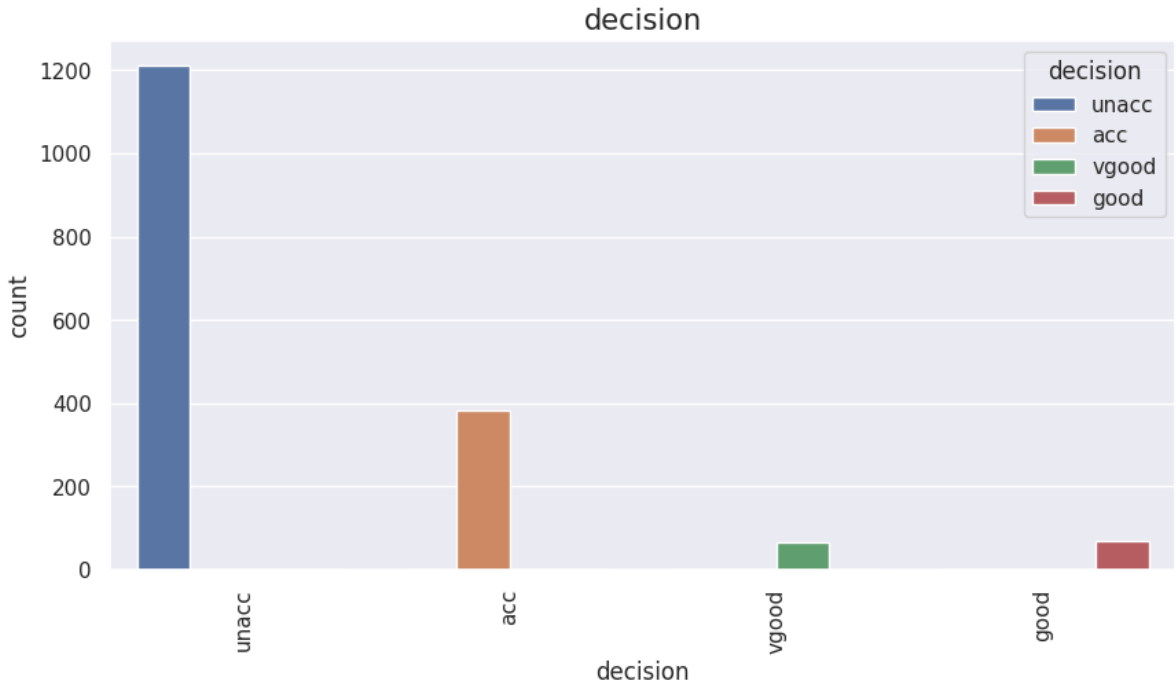

Maintenance cost
decision
unacc
acc
vgood
good
count
0
50
100
150
200
250
300
350
vhigh
high
med
low
Maintenance cost


Number of doors
decision
unacc
acc
vgood
good
count
0
50
100
150
200
250
300
2
3
4
5more
Number of doors


Number of persons
decision
unacc
acc
vgood
good
count
0
100
200
300
400
500
600
2
4
more
Number of persons


lug_boot
decision
unacc
acc
vgood
good
count
0
100
200
300
400
small
med
big
lug_boot

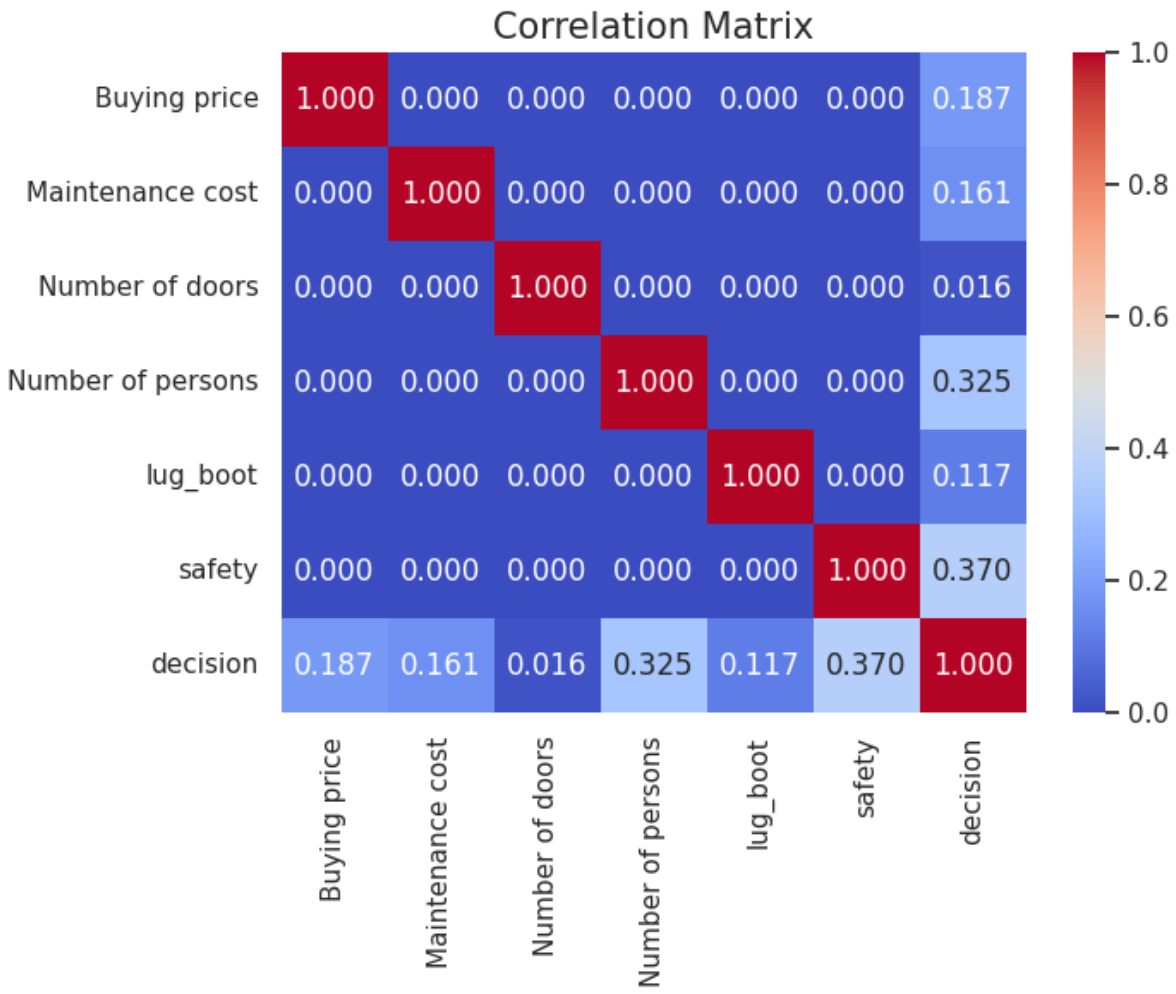

safety
decision
unacc
acc
vgood
good
count
0
100
200
300
400
500
600
low
med
high
safety



Cramer's V for correlation estimation

In [ ]:

```python
def cramers_v(confusion_matrix):
    chi2 = chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    phi2 = chi2 / n
    r, k = confusion_matrix.shape
    phi2corr = max(0, phi2 - ((k-1)*(r-1))/(n-1))
    rcorr = r - ((r-1)**2)/(n-1)
    kcorr = k - ((k-1)**2)/(n-1)
    return np.sqrt(phi2corr / min((kcorr-1), (rcorr-1)))

# Calculate correlation matrix for all features (including numerical and categorica
correlation_matrix = pd.DataFrame(index=df.columns, columns=df.columns, dtype=np.fl

for feature1 in df.columns:
    for feature2 in df.columns:
        if df[feature1].dtype == 'object' and df[feature2].dtype == 'object':
            # Calculate Cramér's V for two categorical variables
            conf = pd.crosstab(df[feature1], df[feature2])
            correlation_matrix.loc[feature1, feature2] = cramers_v(conf.values)

# Visualize the correlation matrix using Seaborn heatmap
plt.figure(figsize=(7,5))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".3f")
plt.title("Correlation Matrix",fontsize=15)
plt.show()
```



Independent features are almost uncorrelated. And the Decision feature is clearly dependent on the other features.

Training part

In [ ]:
```
#drop the target
y=df.pop('decision')
X=copy.deepcopy(df)
#splitting the data
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=42,st
```

In [ ]:
```
X_train
```

Out[ ]:

	Buying price	Maintenance cost	Number of doors	Number of persons	lug_boot	safety
101	vhigh	vhigh	5more	more	small	high
844	high	low	5more	2	big	med
1361	low	vhigh	4	4	small	high
1584	low	med	4	more	small	low
566	high	high	2	more	big	high
...	...	...	...	...	...	...
1209	med	low	2	more	med	low
596	high	high	4	2	small	high
814	high	low	4	2	med	med
1196	med	low	2	2	big	high
972	med	high	2	2	small	low

1209 rows × 6 columns

Apply gridsearch crossvalidation to get the best hyper parameters

In [ ]:
```
pipe=Pipeline(steps=[('ohe',OneHotEncoder(handle_unknown="ignore",sparse_output=Fal
params={'classifier__n_estimators':[10,20,30,50],'classifier__min_samples_leaf':[1,
G=GridSearchCV(pipe,param_grid=params,n_jobs=-1,cv=10,verbose=2)
G.fit(X_train,y_train)
print(G.best_score_)
```

```
Fitting 10 folds for each of 288 candidates, totalling 2880 fits
0.9809779614325068
```

In [ ]:
```
G.best_params_
```

Out[ ]:
```
{'classifier__max_features': None,
 'classifier__max_leaf_nodes': None,
 'classifier__min_samples_leaf': 1,
 'classifier__n_estimators': 30}
```

In [ ]:
```
X_num_test=X_test.apply(LabelEncoder().fit_transform)
yhat_test=G.best_estimator_.predict(X_test)
```

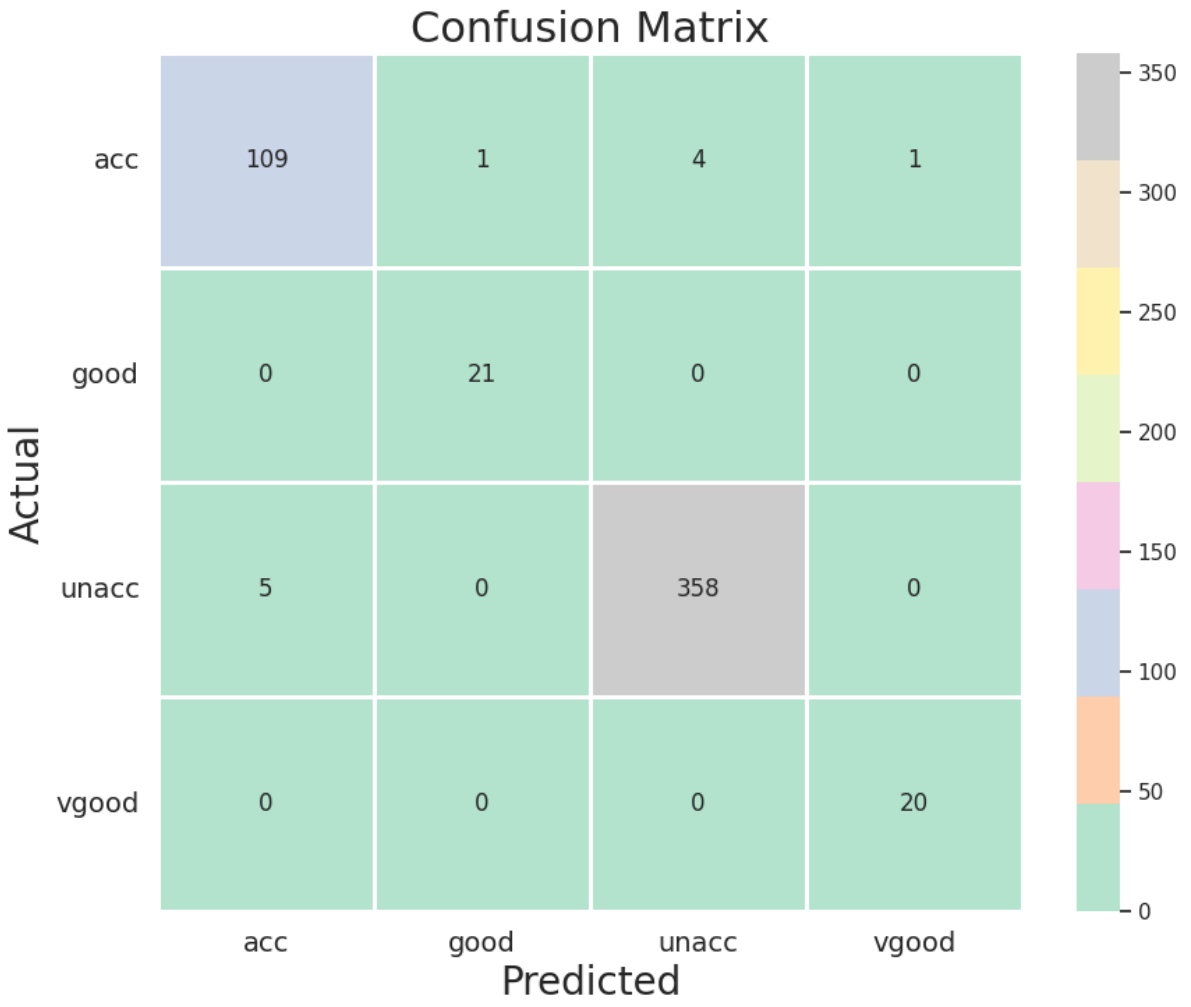
Model Evaluation

In [ ]:
```
print(classification_report(y_test,yhat_test))
```

	precision	recall	f1-score	support
acc	0.96	0.95	0.95	115
good	0.95	1.00	0.98	21
unacc	0.99	0.99	0.99	363
vgood	0.95	1.00	0.98	20
accuracy			0.98	519
macro avg	0.96	0.98	0.97	519
weighted avg	0.98	0.98	0.98	519

Confusion Matrix

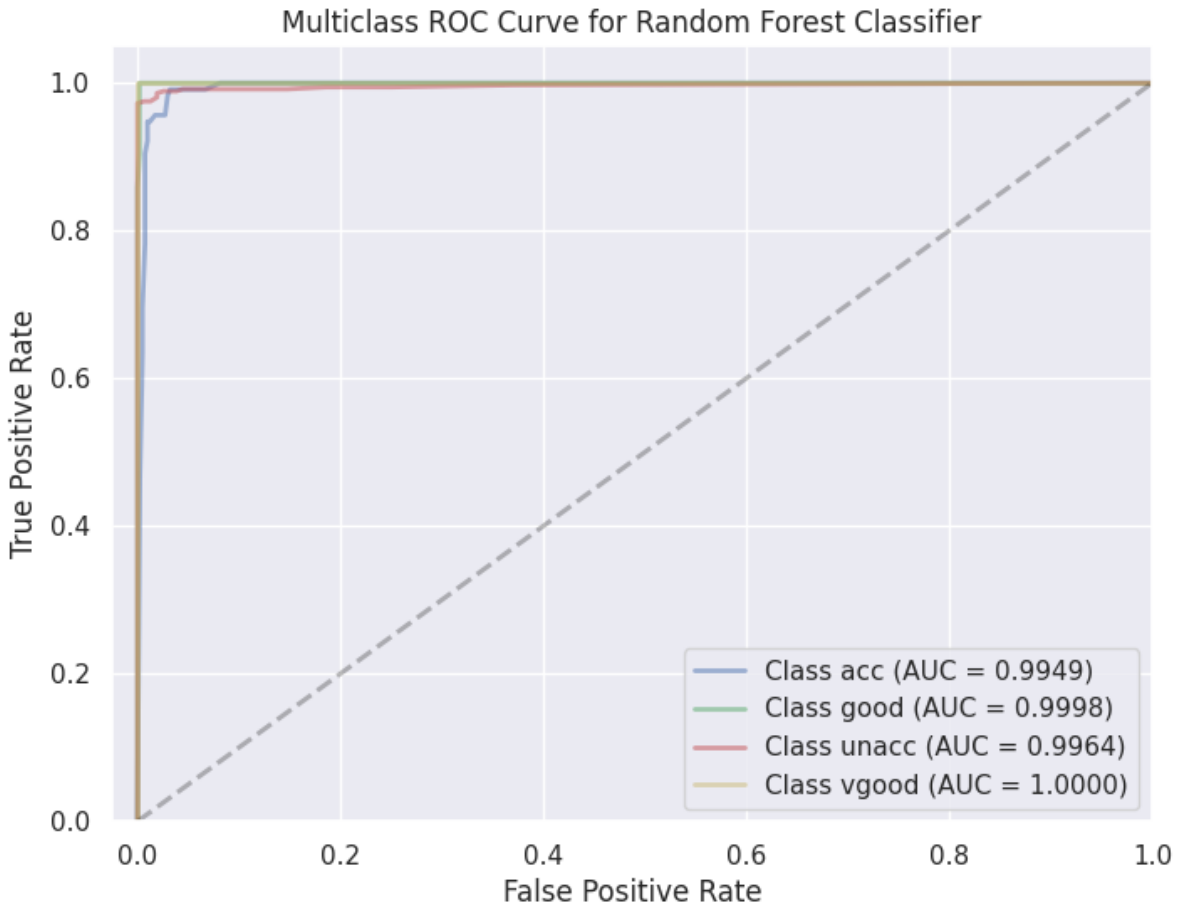
In [ ]:
```
labels=np.unique(y_test)
CM=confusion_matrix(y_test,yhat_test)
plt.figure(figsize=(10,8))
plt.title("Confusion Matrix",fontsize=22)
sns.heatmap(CM,annot=True,cmap='Pastel2',fmt='0.3g',linewidths=1.0)
plt.yticks(np.arange(len(labels))+0.5,labels,fontsize=14,rotation='horizontal')
plt.xticks(np.arange(len(labels))+0.5,labels,fontsize=14)
plt.xlabel("Predicted",fontsize=20)
plt.ylabel("Actual",fontsize=20)
plt.savefig("Conf.png")
plt.show()
```



ROC curve

In [ ]:
```
oh_test=pd.get_dummies(y_test)
oh_pred=G.best_estimator_.predict_proba(X_test)
```

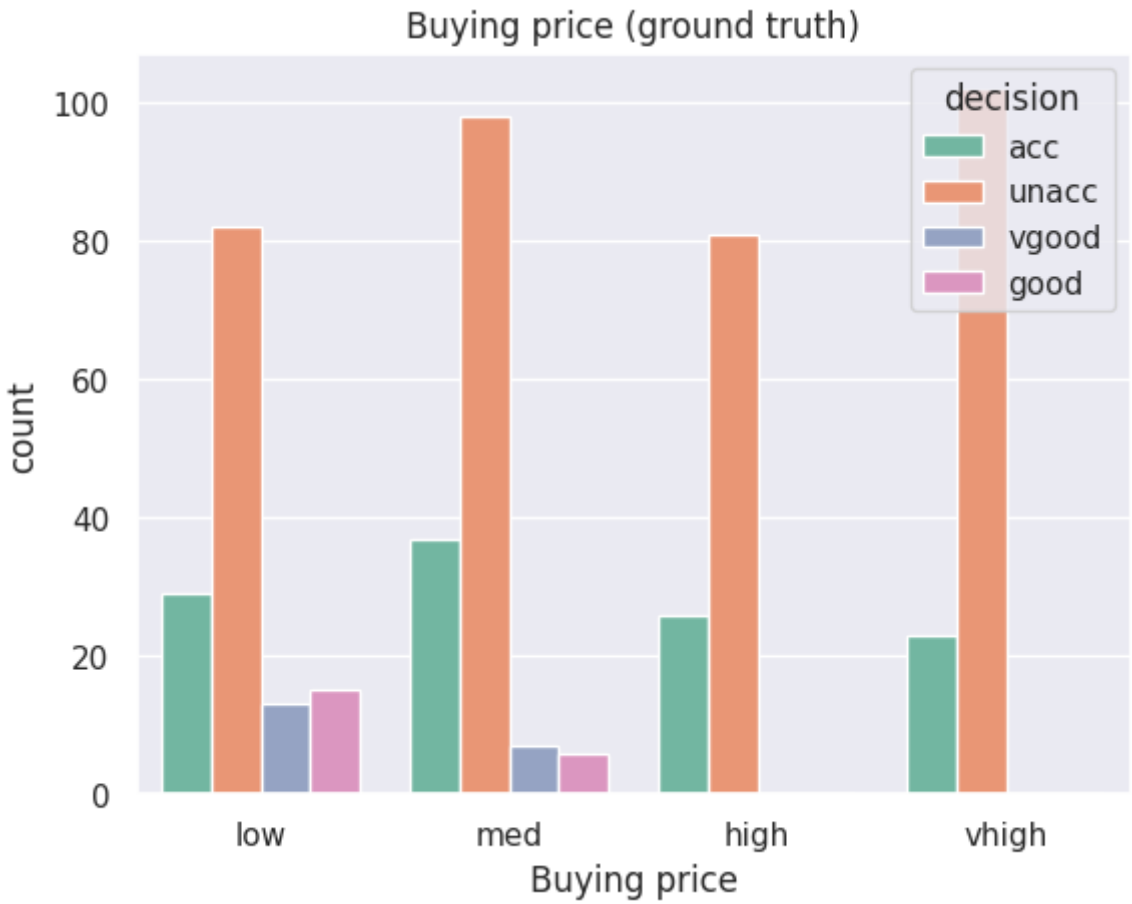
In [ ]:
```
fpr = dict()
tpr = dict()
roc_auc = dict()
num_classes=oh_test.columns
for i in range(len(num_classes)):
    fpr[i], tpr[i], _ = roc_curve(oh_test.loc[:, num_classes[i]], oh_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot the ROC curves for each class
plt.figure(figsize=(8, 6))
colors = ['b', 'g', 'r','y']  # Use appropriate colors based on the number of class
for k,(i, color) in enumerate(zip(num_classes, colors)):
    plt.plot(fpr[k], tpr[k], color=color, lw=2, label=f'Class {i} (AUC = {roc_auc[k

plt.plot([0, 1], [0, 1], 'k--', lw=2,alpha=0.3)
plt.xlim([-0.025, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass ROC Curve for Random Forest Classifier')
plt.legend(loc='lower right')
plt.savefig("ROC.png")
plt.show()
```

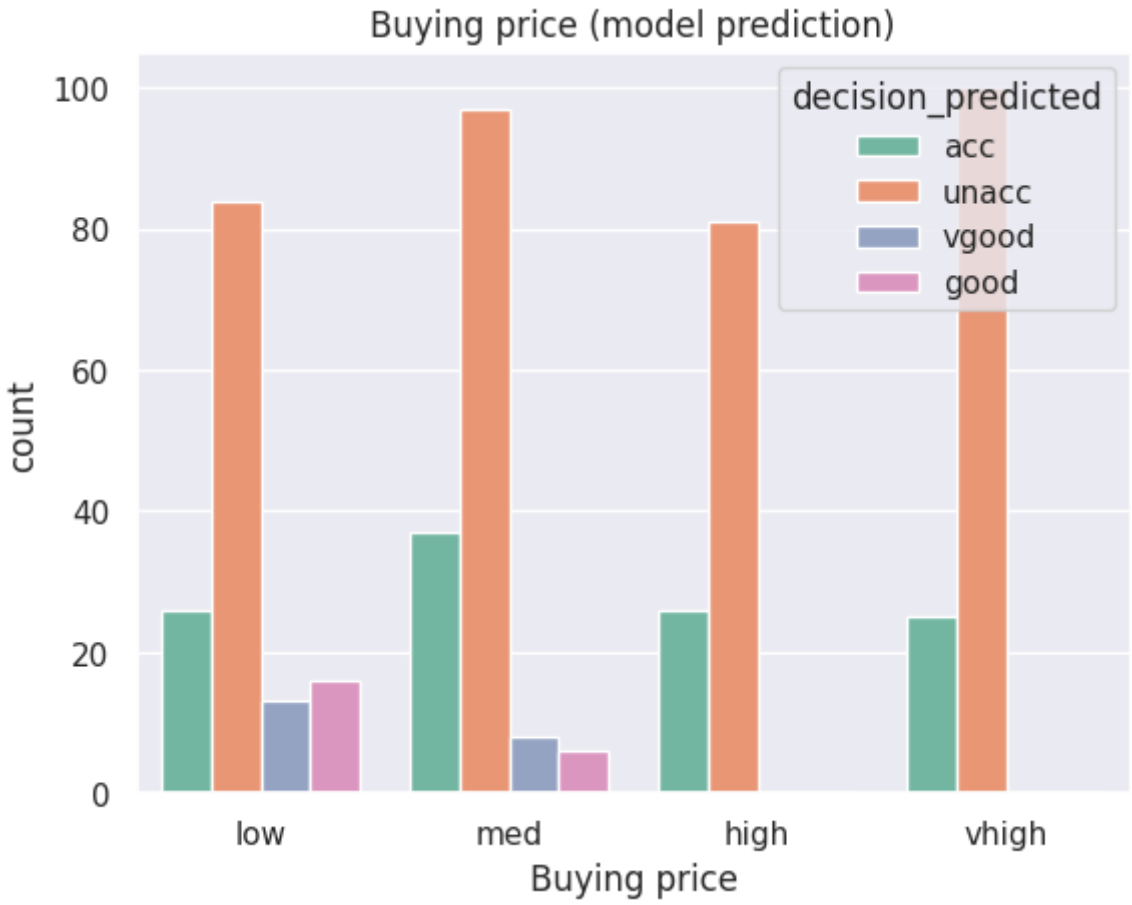


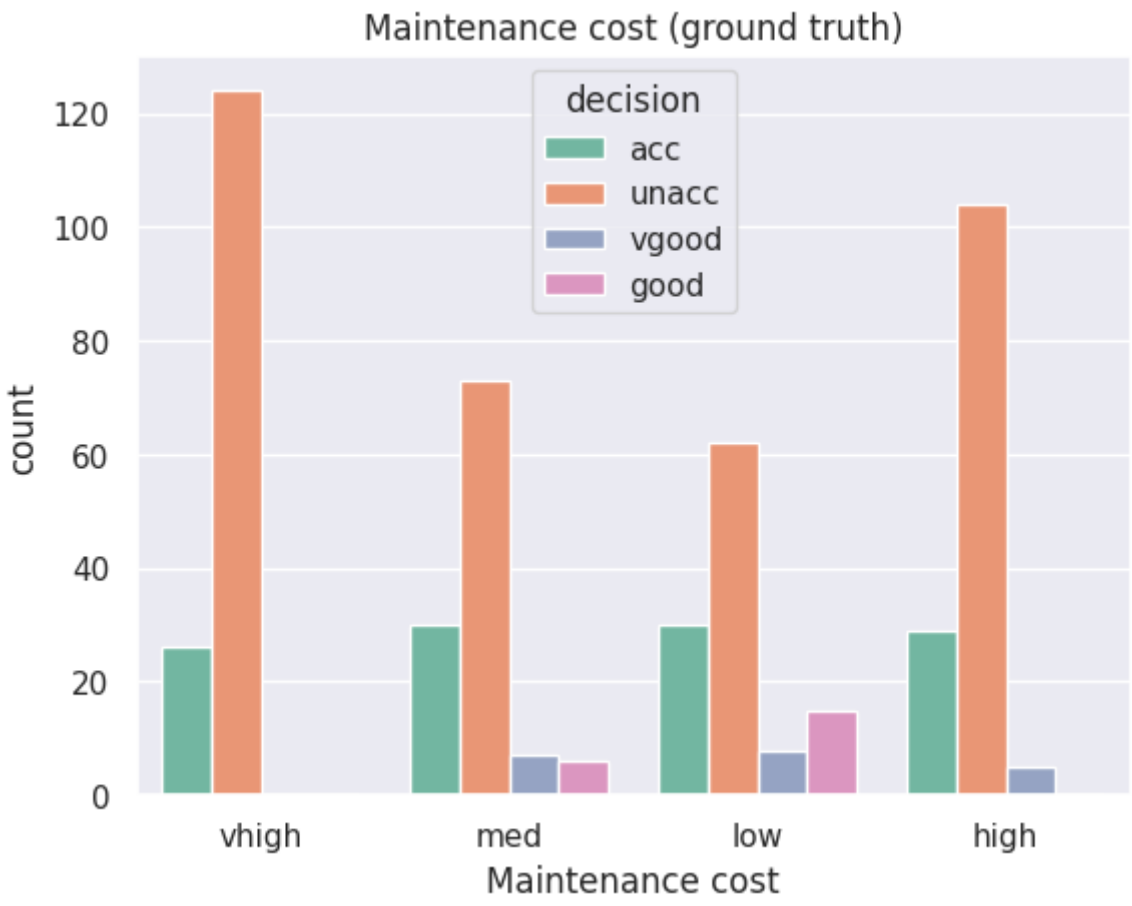
Visualization on the test data

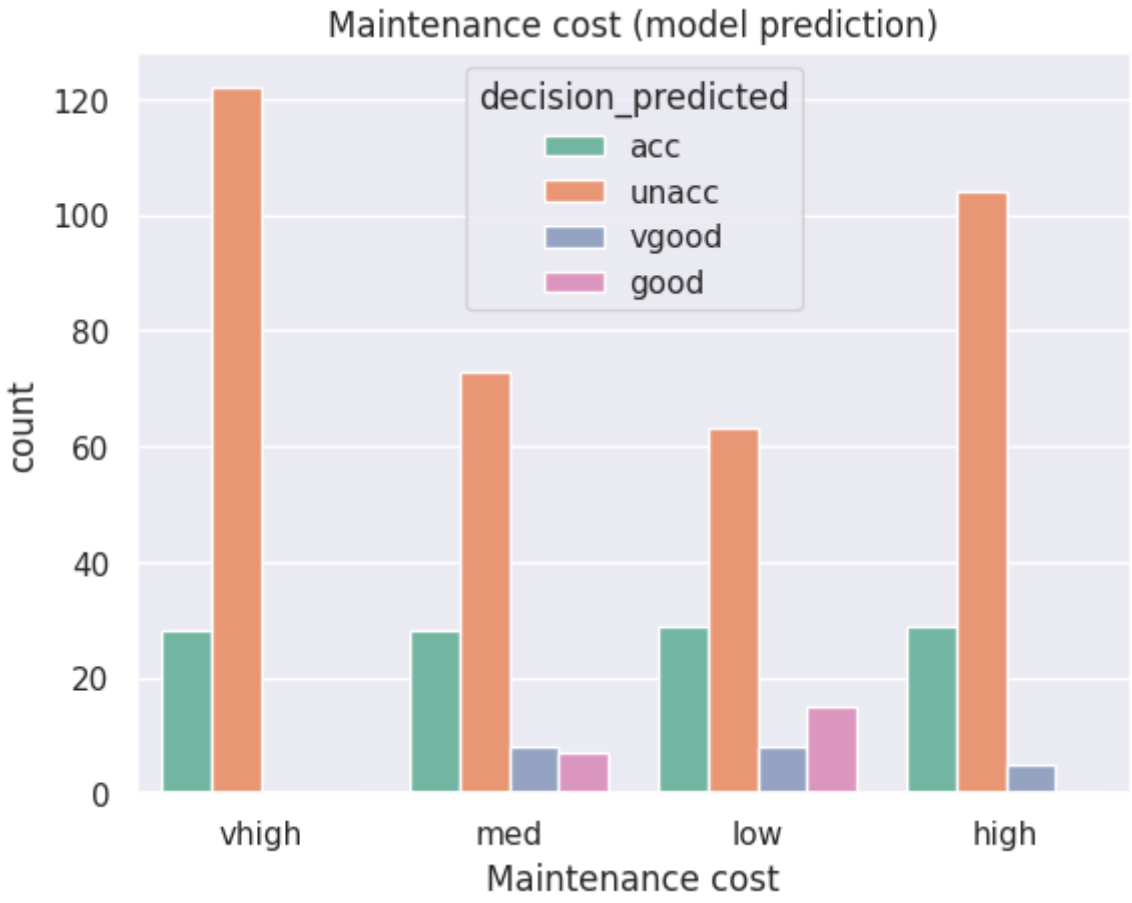
In [ ]:
```
X_test1=copy.deepcopy(X_test)
X_test2=copy.deepcopy(X_test)
X_test1['decision']=y_test
```

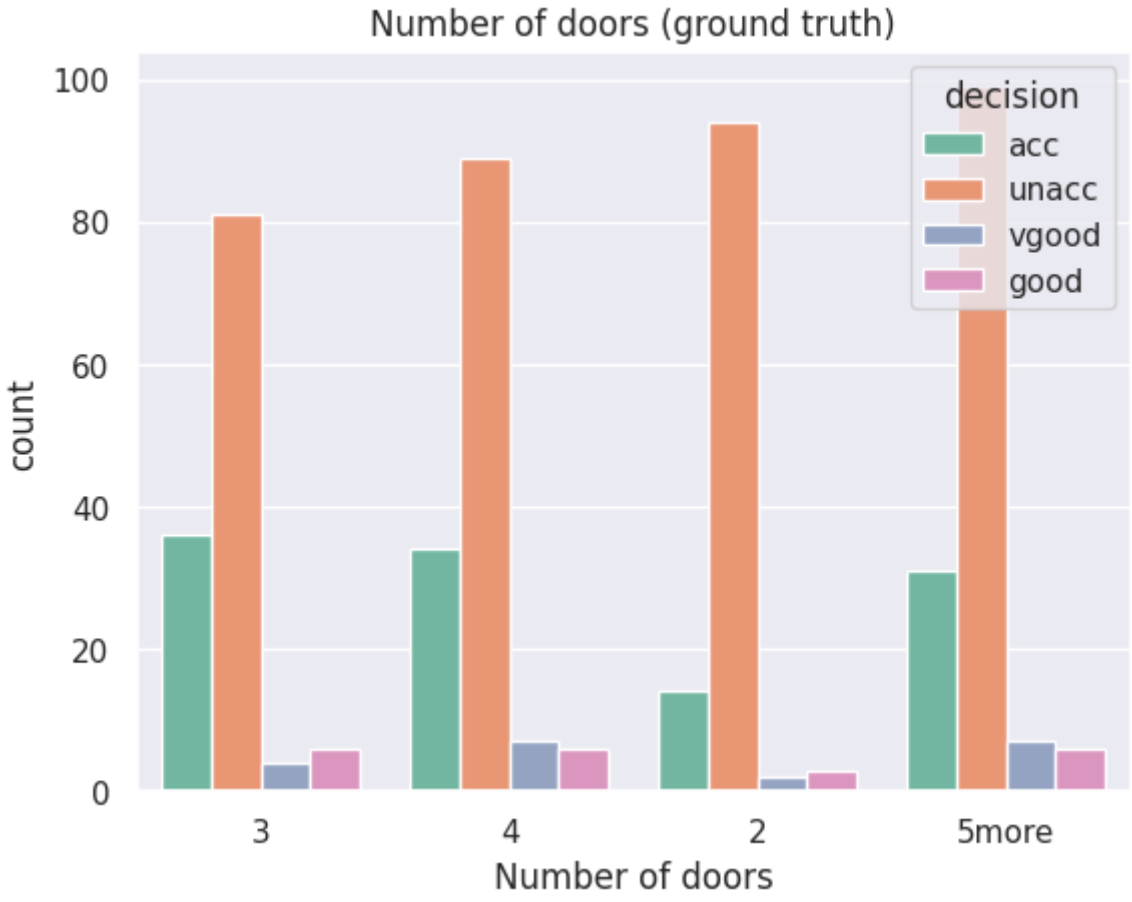
```
X_test2['decision_predicted']=yhat_test
cols=['Buying price','Maintenance cost','Number of doors','Number of persons','lug_
for cat in cols:
  plt.title(f"{cat} (ground truth)")
  sns.countplot(x=cat,data=X_test1,hue='decision',palette='Set2')
  plt.savefig(f"test_{cat}_tru.png",bbox_inches = 'tight')
  plt.show()
  plt.title(f"{cat} (model prediction)")
  sns.countplot(x=cat,data=X_test2,hue='decision_predicted',palette='Set2')
  plt.savefig(f"test_{cat}_pred.png",bbox_inches = 'tight')
  plt.show()
```

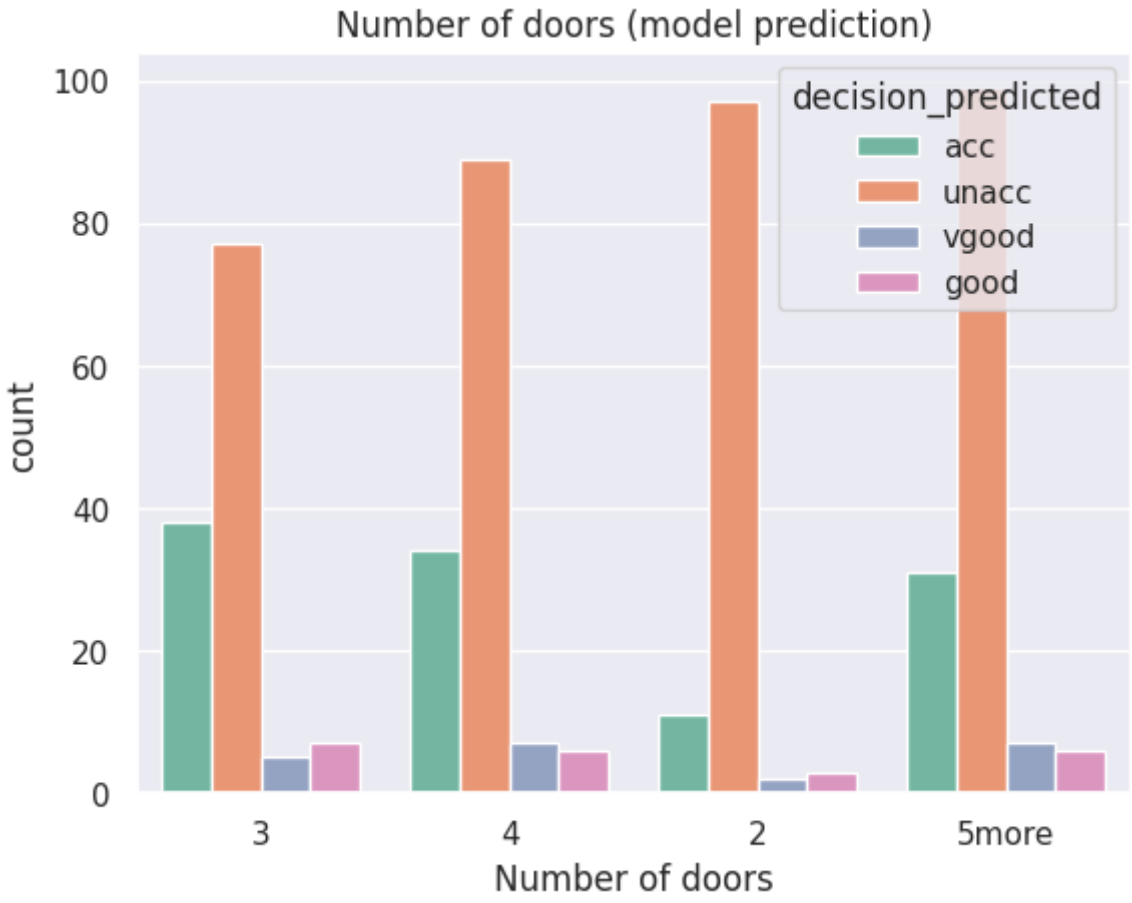


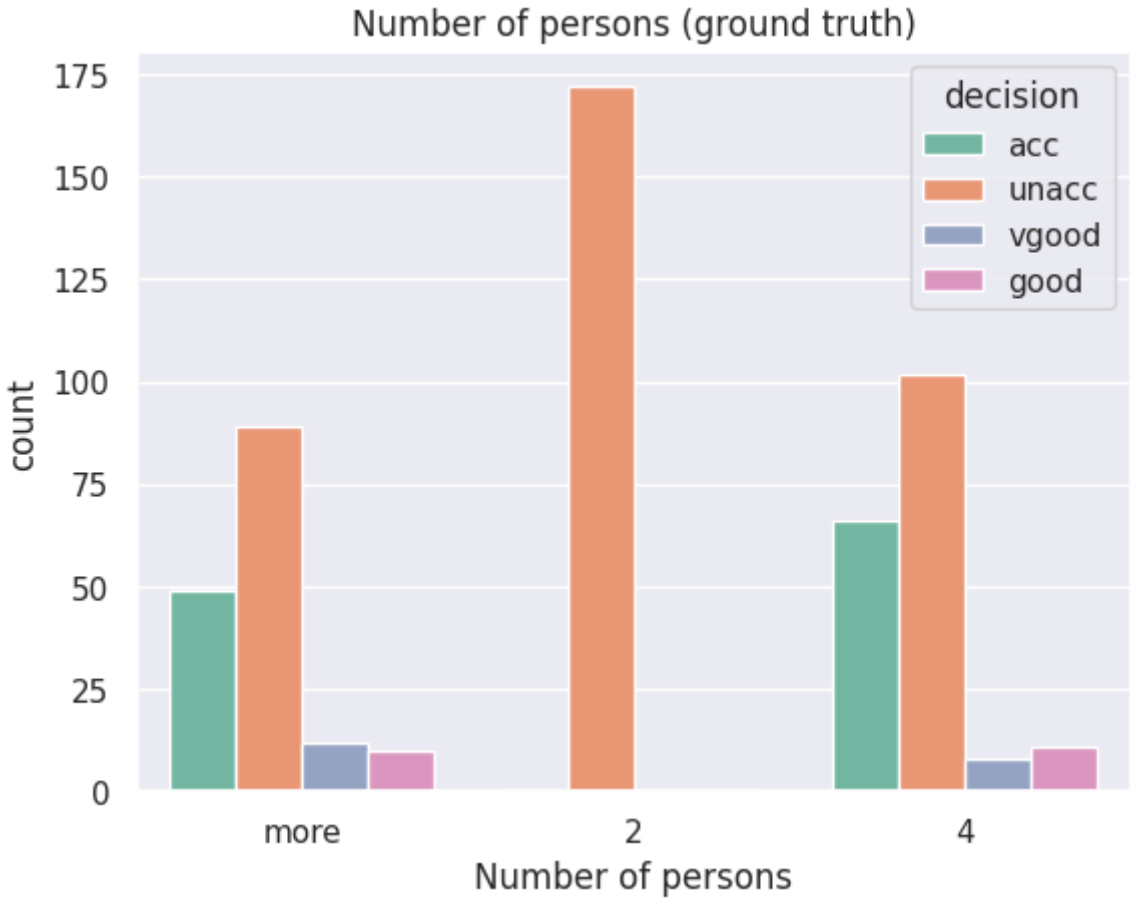

Buying price (model prediction)
decision_predicted
acc
unacc
vgood
good
count
0
20
40
60
80
100
low
med
high
vhigh
Buying price


Maintenance cost (ground truth)
decision
acc
unacc
vgood
good
count
0
20
40
60
80
100
120
vhigh
med
low
high
Maintenance cost

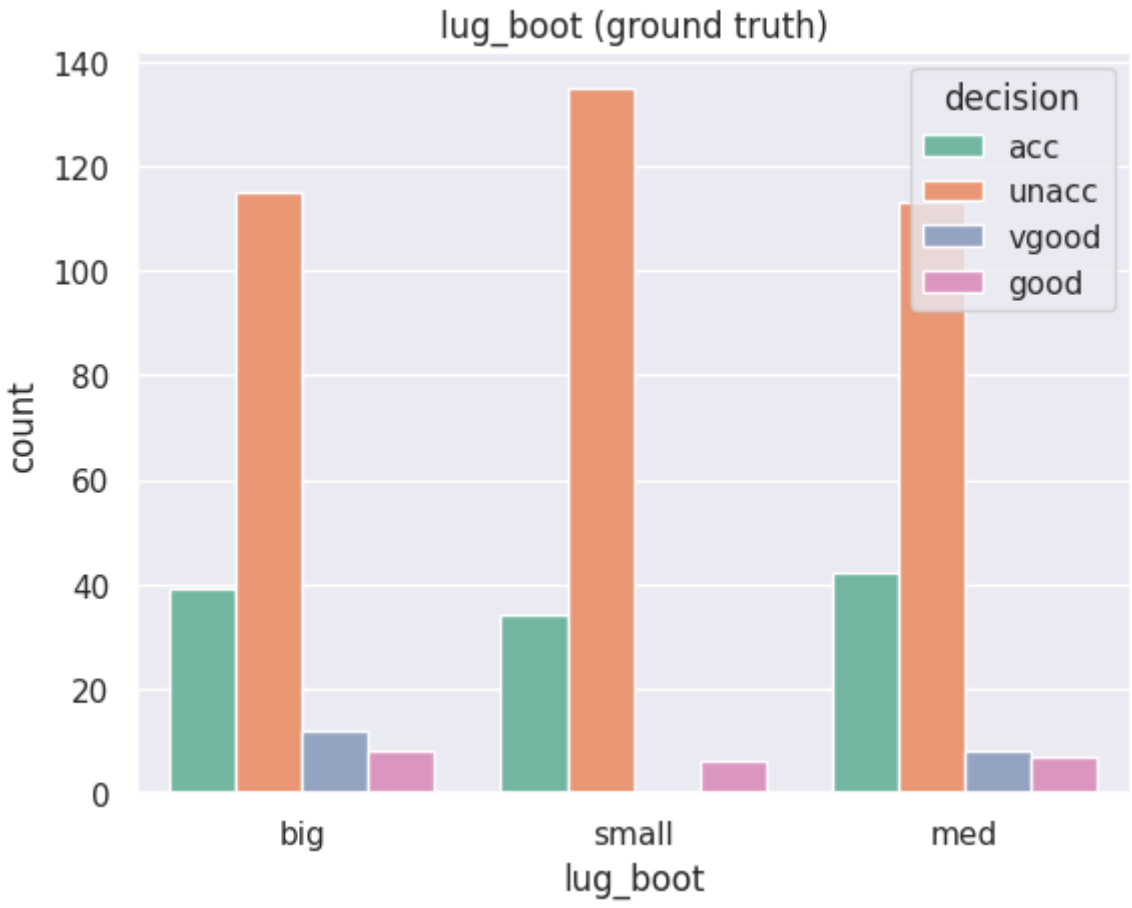

Maintenance cost (model prediction)
decision_predicted
acc
unacc
vgood
good
count
0
20
40
60
80
100
120
vhigh
med
low
high
Maintenance cost

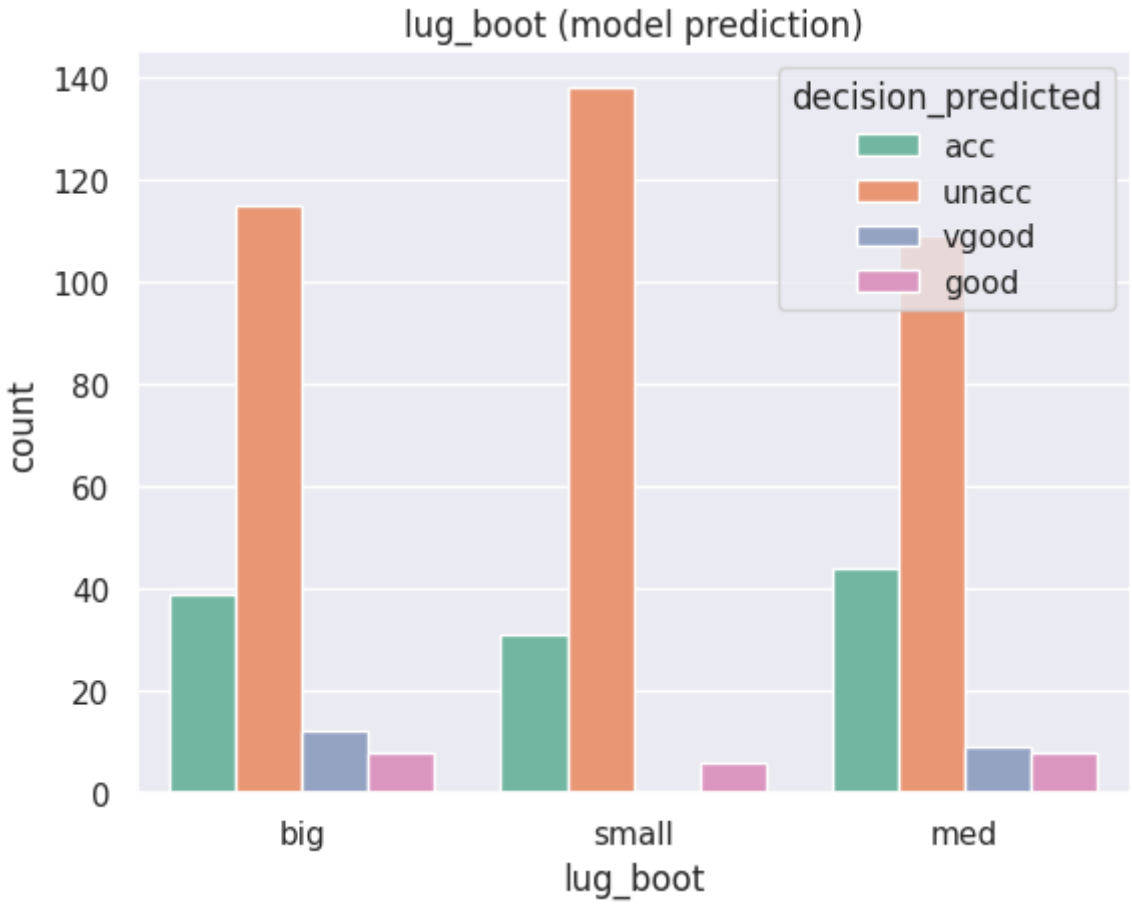

Number of doors (ground truth)
decision
acc
unacc
vgood
good
count
0
20
40
60
80
100
3
4
2
5more
Number of doors

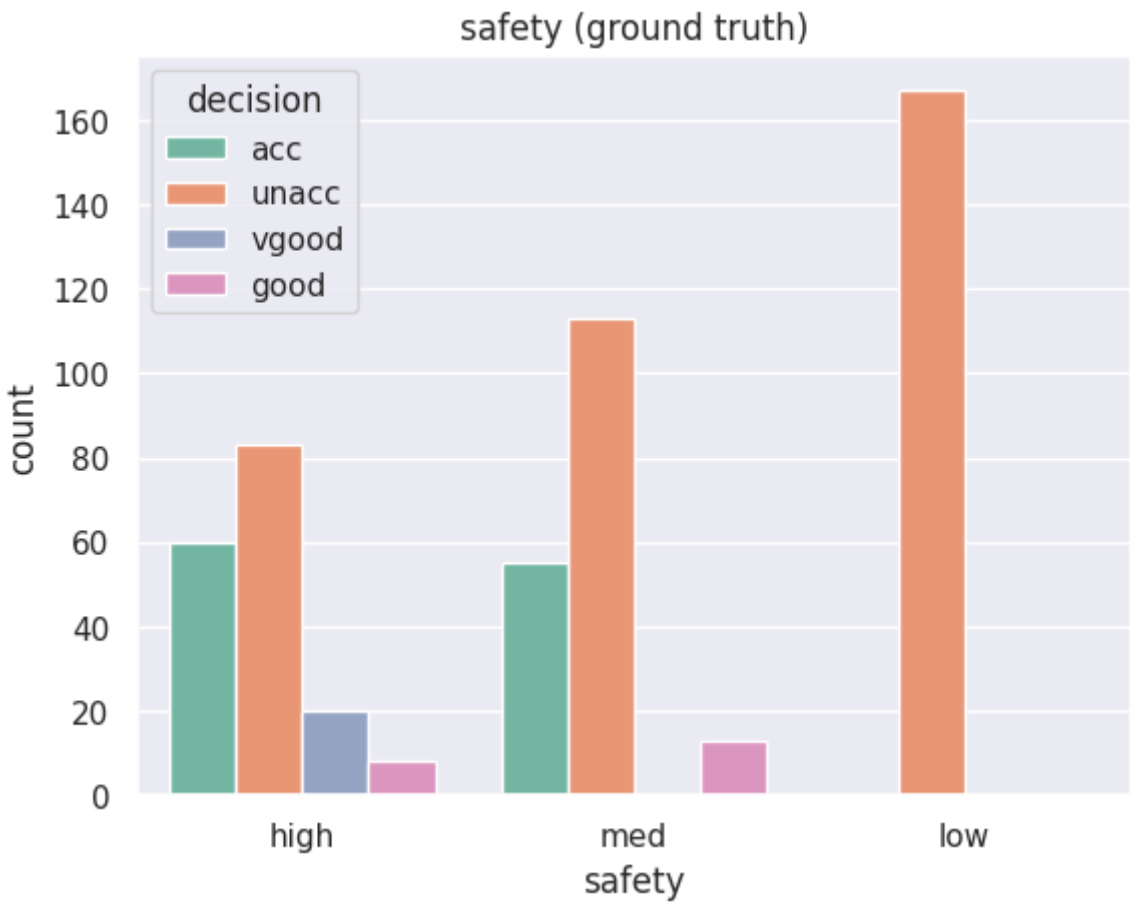

Number of doors (model prediction)
decision_predicted
acc
unacc
vgood
good
count
100
80
60
40
20
0
3
4
2
5more
Number of doors


Number of persons (ground truth)
decision
acc
unacc
vgood
good
count
175
150
125
100
75
50
25
0
more
2
4
Number of persons


Number of persons (model prediction)
decision_predicted
acc
unacc
vgood
good
count
175
150
125
100
75
50
25
0
more
2
4
Number of persons


lug_boot (ground truth)
decision
acc
unacc
vgood
good
count
140
120
100
80
60
40
20
0
big
small
med
lug_boot


lug_boot (model prediction)
decision_predicted
acc
unacc
vgood
good
count
0
20
40
60
80
100
120
140
big
small
med
lug_boot


safety (ground truth)
decision
acc
unacc
vgood
good
count
0
20
40
60
80
100
120
140
160
high
med
low
safety


safety (model prediction)
decision_predicted
acc
unacc
vgood
good
160
140
120
100
80
60
40
20
0
count
high
med
low
safety

End of Code

DAL Assignment Support Vector Machine (revised)

Importing the libraries

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import KNNImputer
from sklearn.ensemble import BaggingClassifier
from sklearn.preprocessing import MinMaxScaler
#from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report,roc_curve,auc,confusion_matrix
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
import copy
```

Load the data

In [ ]:
```
Data_train=pd.read_excel(r'/content/drive/MyDrive/DAL dataset/Assignment 6/pulsar_d
```

In [ ]:
```
Data_train.info()# overall information of the data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12528 entries, 0 to 12527
Data columns (total 9 columns):
 #   Column                                         Non-Null Count  Dtype  
---  ------                                         --------------  -----  
 0    Mean of the integrated profile                12528 non-null  float64
 1    Standard deviation of the integrated profile  12528 non-null  float64
 2    Excess kurtosis of the integrated profile     10793 non-null  float64
 3    Skewness of the integrated profile            12528 non-null  float64
 4    Mean of the DM-SNR curve                      12528 non-null  float64
 5    Standard deviation of the DM-SNR curve        11350 non-null  float64
 6    Excess kurtosis of the DM-SNR curve           12528 non-null  float64
 7    Skewness of the DM-SNR curve                  11903 non-null  float64
 8   target_class                                   12528 non-null  int64  
dtypes: float64(8), int64(1)
memory usage: 881.0 KB
```

In [ ]:
```
# The columns have leading white space so removing them
Data_train.columns = Data_train.columns.str.strip()
```

Get the columns having missing entries

In [ ]:
```
Missing=Data_train.isna().any().values
Missing_cols=Data_train.columns[Missing]
Missing_cols#These are the columns with missing entries
```

Out[ ]:
```
Index(['Excess kurtosis of the integrated profile',
       'Standard deviation of the DM-SNR curve',
       'Skewness of the DM-SNR curve'],
      dtype='object')
```

In [ ]:
```
Data_train['target_class'].value_counts()# So its a binary classifier
```

Out[ ]:
```
0    11375
1     1153
Name: target_class, dtype: int64
```

In [ ]:
```
Data_train.describe()# Statistical details of the data
```

Out[ ]:

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	kur the D
count	12528.000000	12528.000000	10793.000000	12528.000000	12528.000000	11350.000000	12528
mean	111.041841	46.521437	0.478548	1.778431	12.674758	26.351318	8
std	25.672828	6.801077	1.064708	6.208450	29.613230	19.610842	4
min	5.812500	24.772042	-1.738021	-1.791886	0.213211	7.370432	-3
25%	100.871094	42.362222	0.024652	-0.188142	1.910535	14.404353	5
50%	115.183594	46.931022	0.223678	0.203317	2.792642	18.412402	8
75%	127.109375	50.979103	0.473125	0.932374	5.413253	28.337418	10
max	189.734375	91.808628	8.069522	68.101622	222.421405	110.642211	34

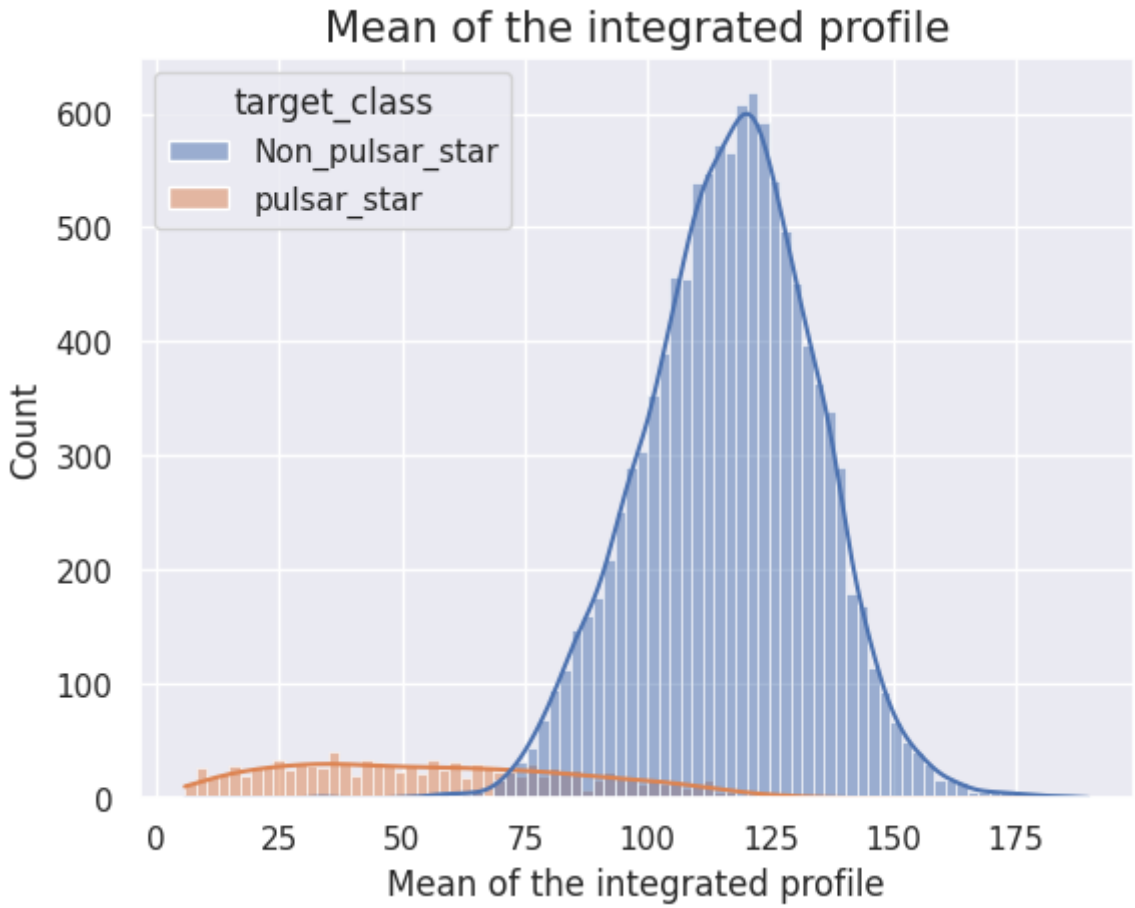
In [ ]:
```
sns.set()
def Plot_dist(data):
  """
  Give a DataFrame
  """
  data=copy.deepcopy(data)
  Columns=data.columns
  data[Columns[-1]].replace(0,'Non_pulsar_star',inplace=True)
  data[Columns[-1]].replace(1,'pulsar_star',inplace=True)
  for col in Columns:
    plt.title(col,fontsize=15)
    sns.histplot(data, x=col, hue=Columns[-1], kde=True)
    plt.show()
```

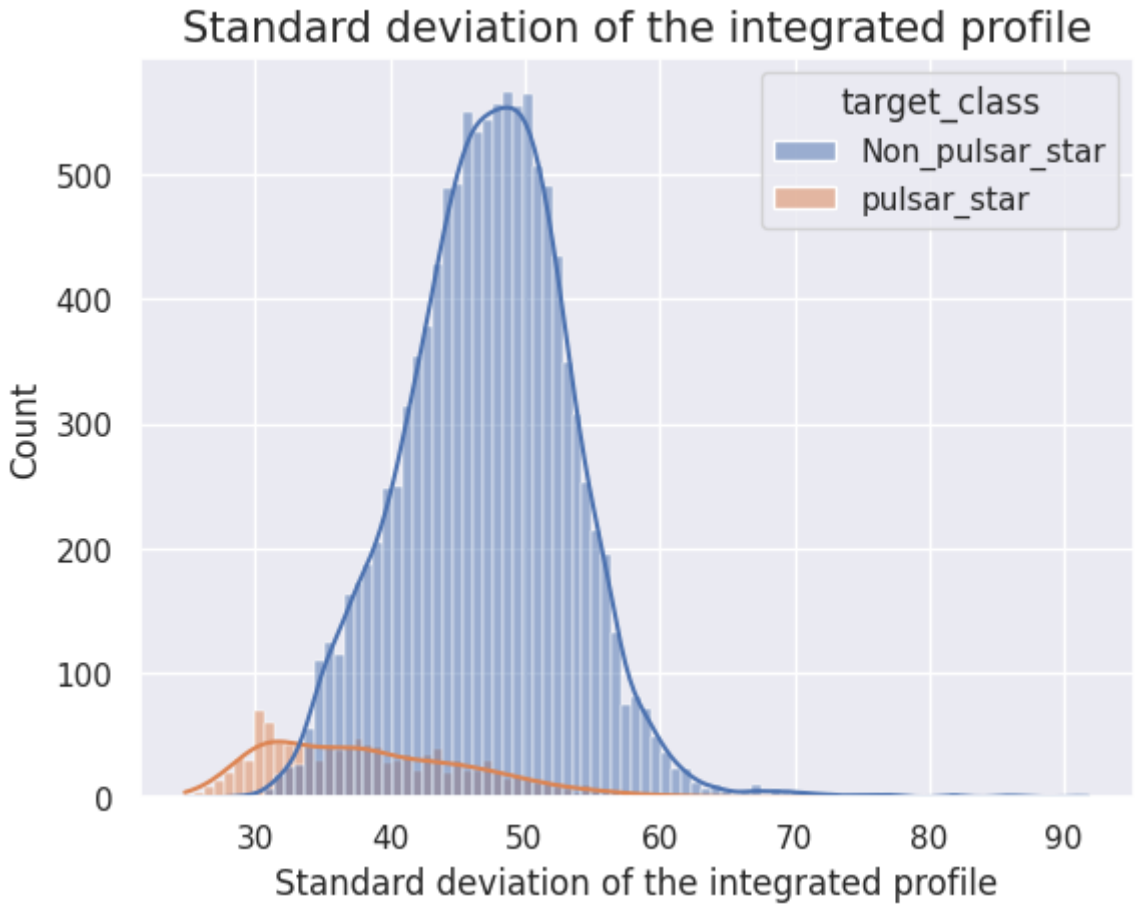
In [ ]:
```
sns.set()
def Plot_kde(data):
  """
  Give a DataFrame
  """
  Columns=data.columns
  for col in Columns:
    plt.title(f'{col} (test set)',fontsize=15)
    sns.kdeplot(data, x=col, hue=Columns[-1],fill=True)
    plt.show()
```

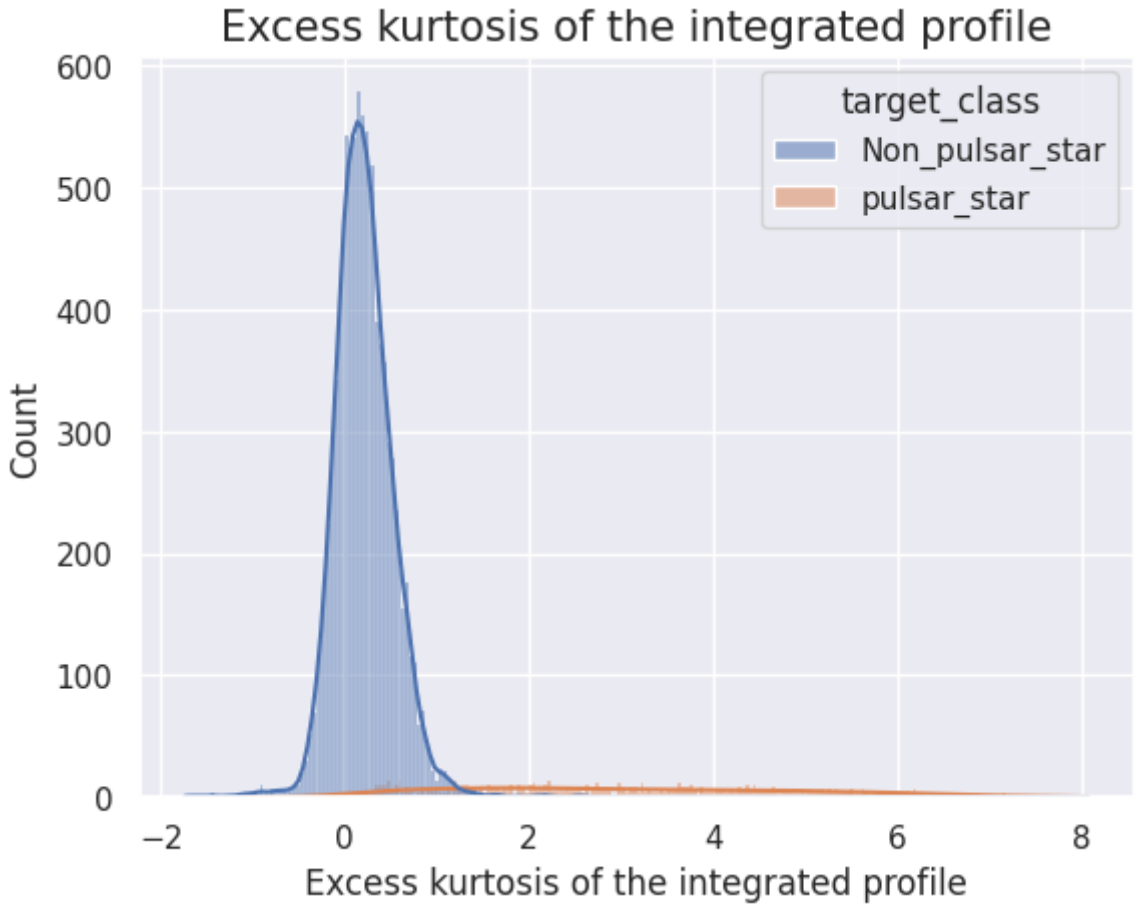
In [ ]:
```
def Correlation_plot(Data):
  cor=Data.corr()
  plt.figure(figsize=(10,8))
```

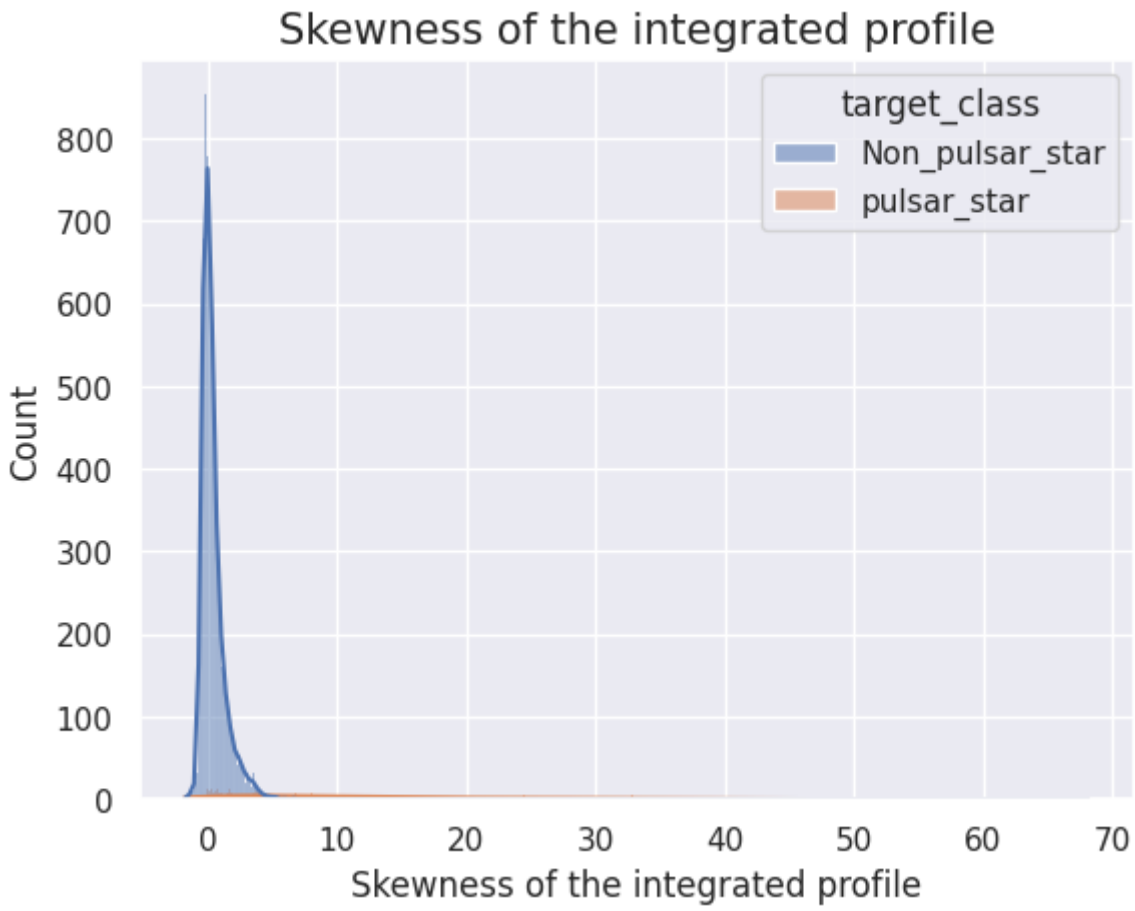
```
  sns.heatmap(cor,annot=True)
  plt.title("Heat map for correlation",fontsize=15)
  plt.show()
```

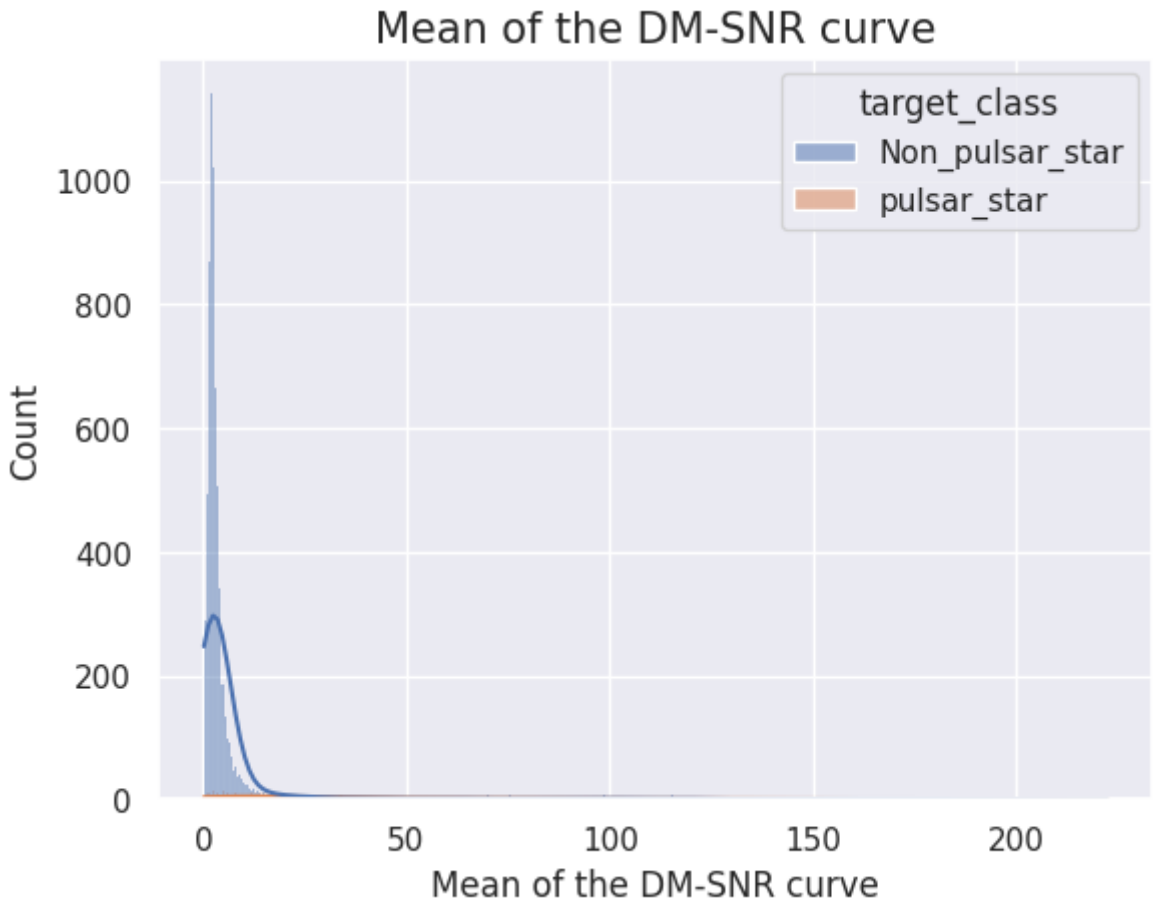
In [ ]:
```
Plot_dist(Data_train)
```

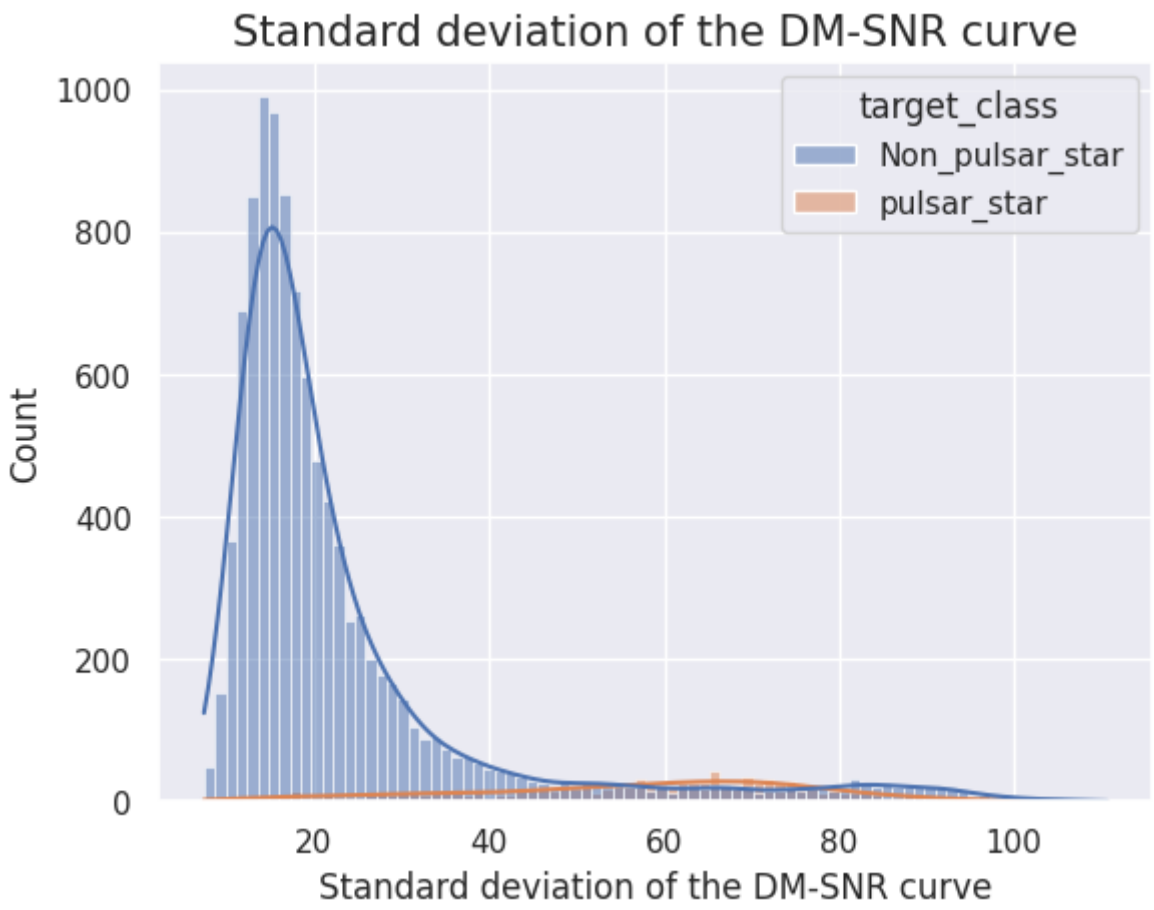


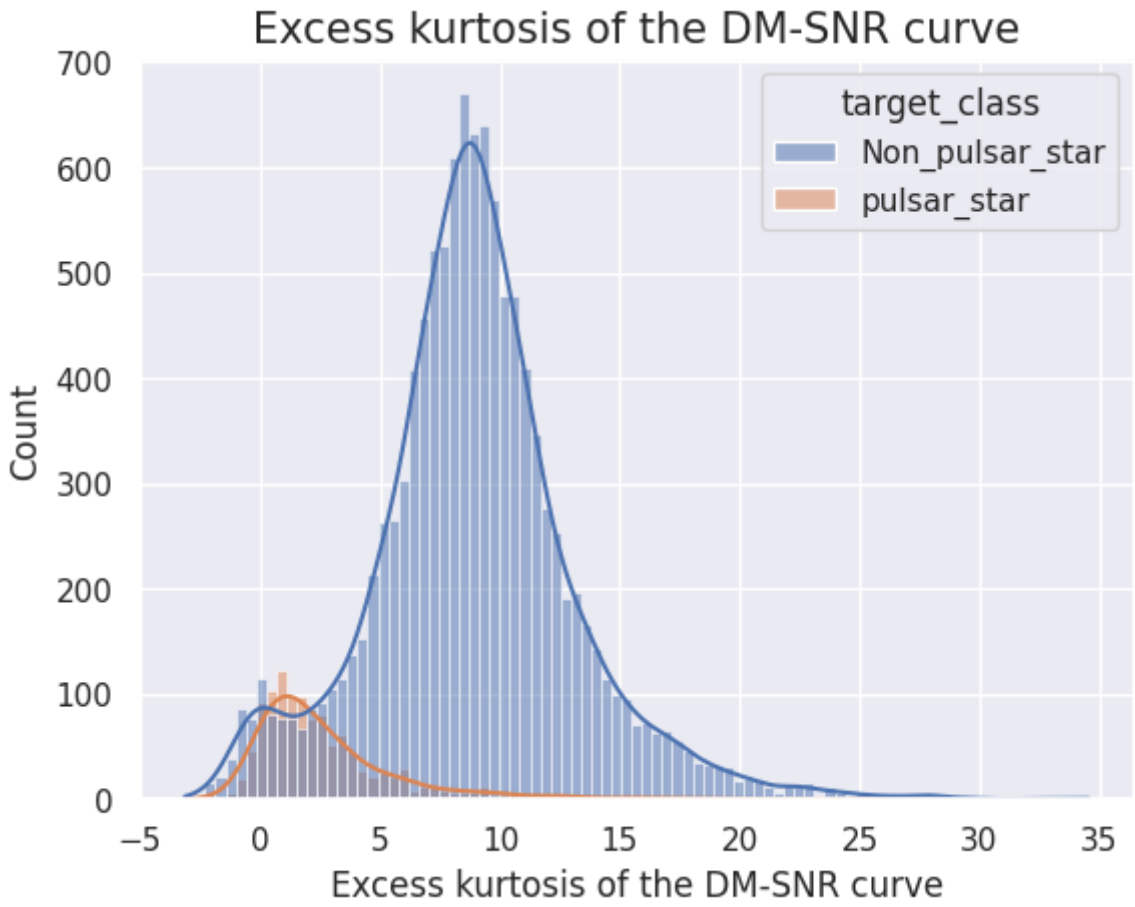


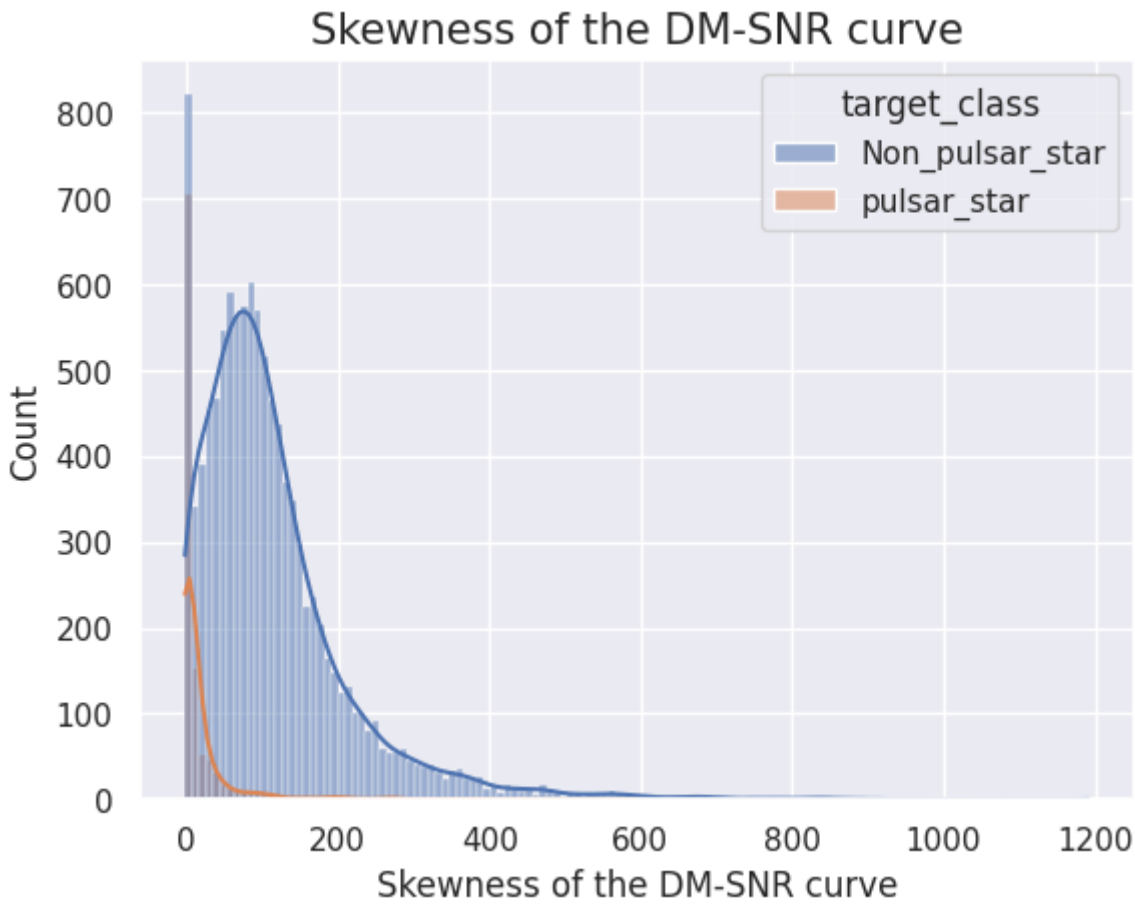

Excess kurtosis of the integrated profile
target_class
Non_pulsar_star
pulsar_star
Count
600
500
400
300
200
100
0
−2
0
2
4
6
8
Excess kurtosis of the integrated profile

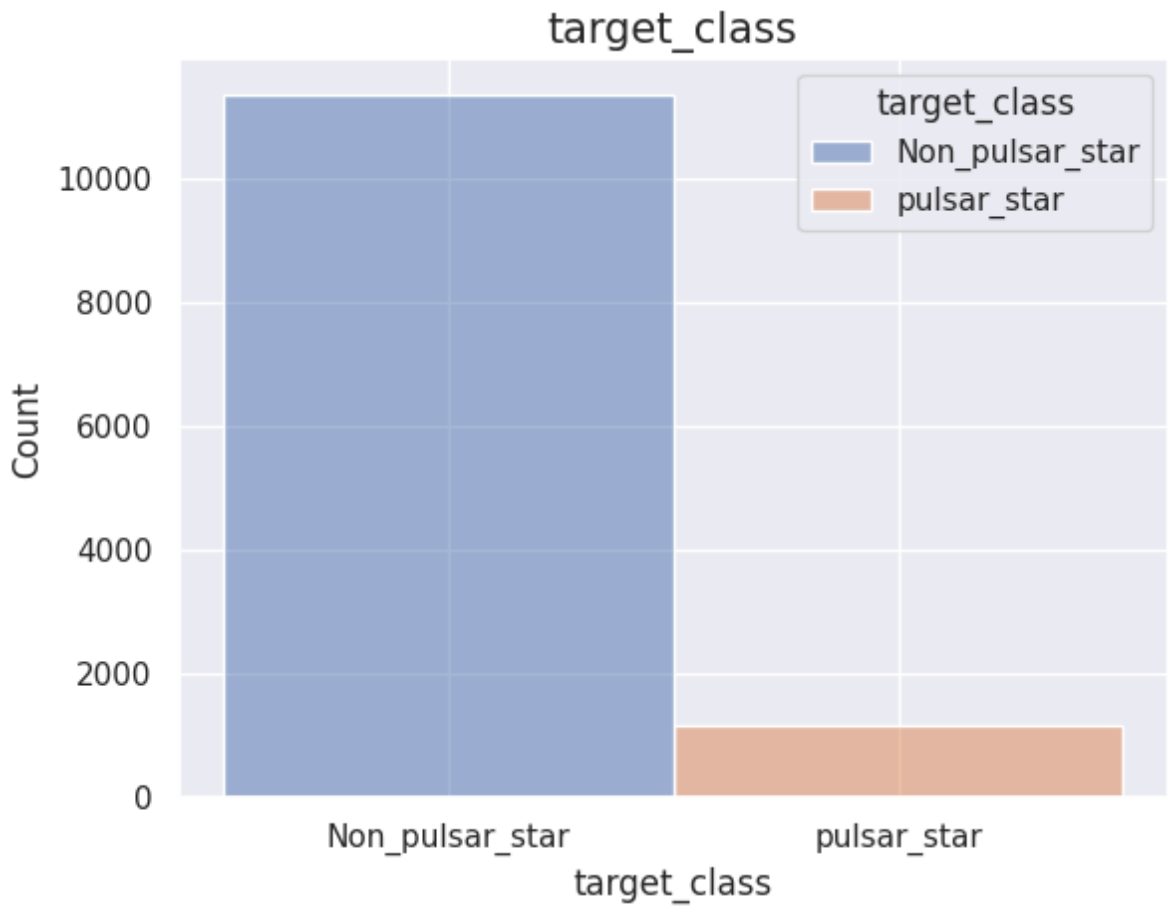

Skewness of the integrated profile
target_class
Non_pulsar_star
pulsar_star
Count
800
700
600
500
400
300
200
100
0
0
10
20
30
40
50
60
70
Skewness of the integrated profile


Mean of the DM-SNR curve
target_class
Non_pulsar_star
pulsar_star
Count
0
200
400
600
800
1000
0
50
100
150
200
Mean of the DM-SNR curve


Standard deviation of the DM-SNR curve
target_class
Non_pulsar_star
pulsar_star
Count
0
200
400
600
800
1000
20
40
60
80
100
Standard deviation of the DM-SNR curve


Excess kurtosis of the DM-SNR curve
target_class
Non_pulsar_star
pulsar_star
Count
0
100
200
300
400
500
600
700
−5
0
5
10
15
20
25
30
35
Excess kurtosis of the DM-SNR curve


Skewness of the DM-SNR curve
target_class
Non_pulsar_star
pulsar_star
Count
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200
Skewness of the DM-SNR curve



From the above plots it is clear most of the features are skewed in nature and the target column is highly imbalanced

In [ ]:
```
# All the three missing entry columns have skewed distribution
# For skewed data median should be used for imputation
# We will perform KNN imputation
```

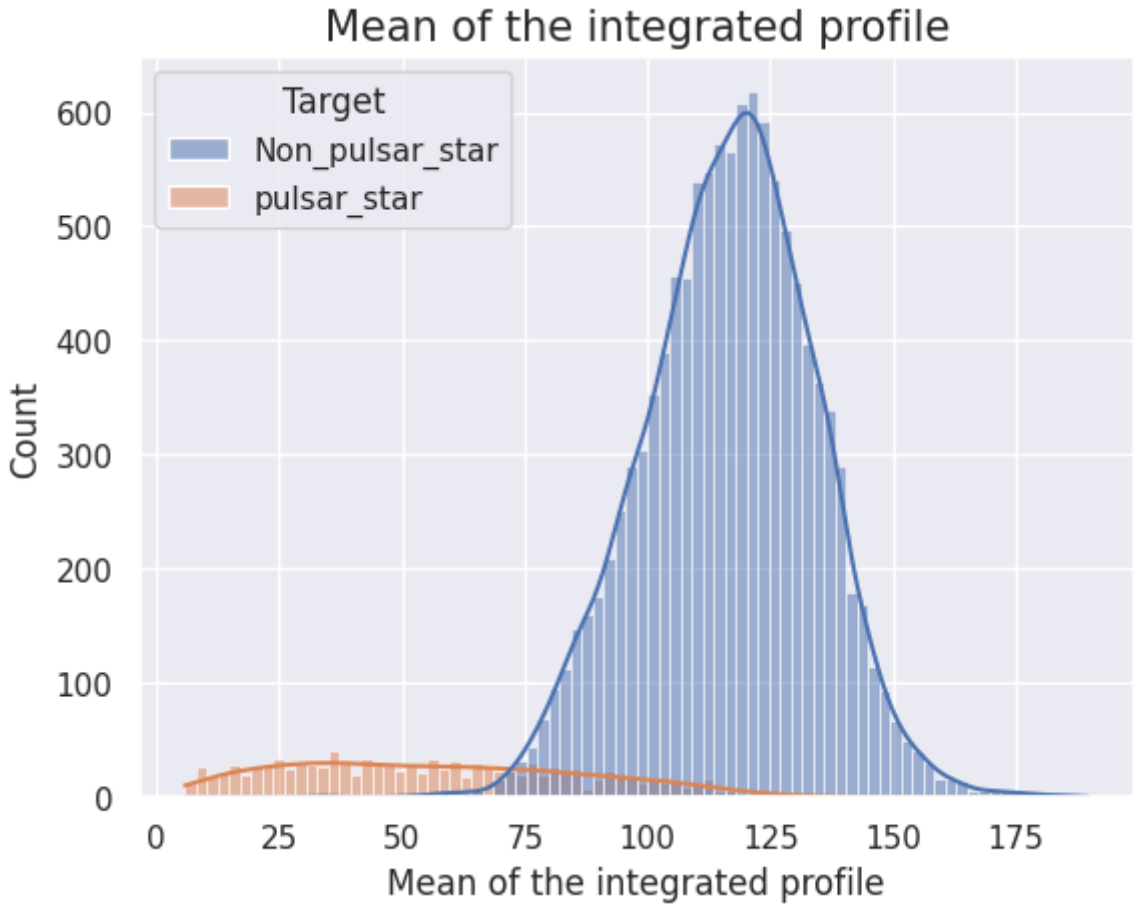
In [ ]:
```
# Drop the target column and columns with missing entries
Target=Data_train.pop('target_class')
```

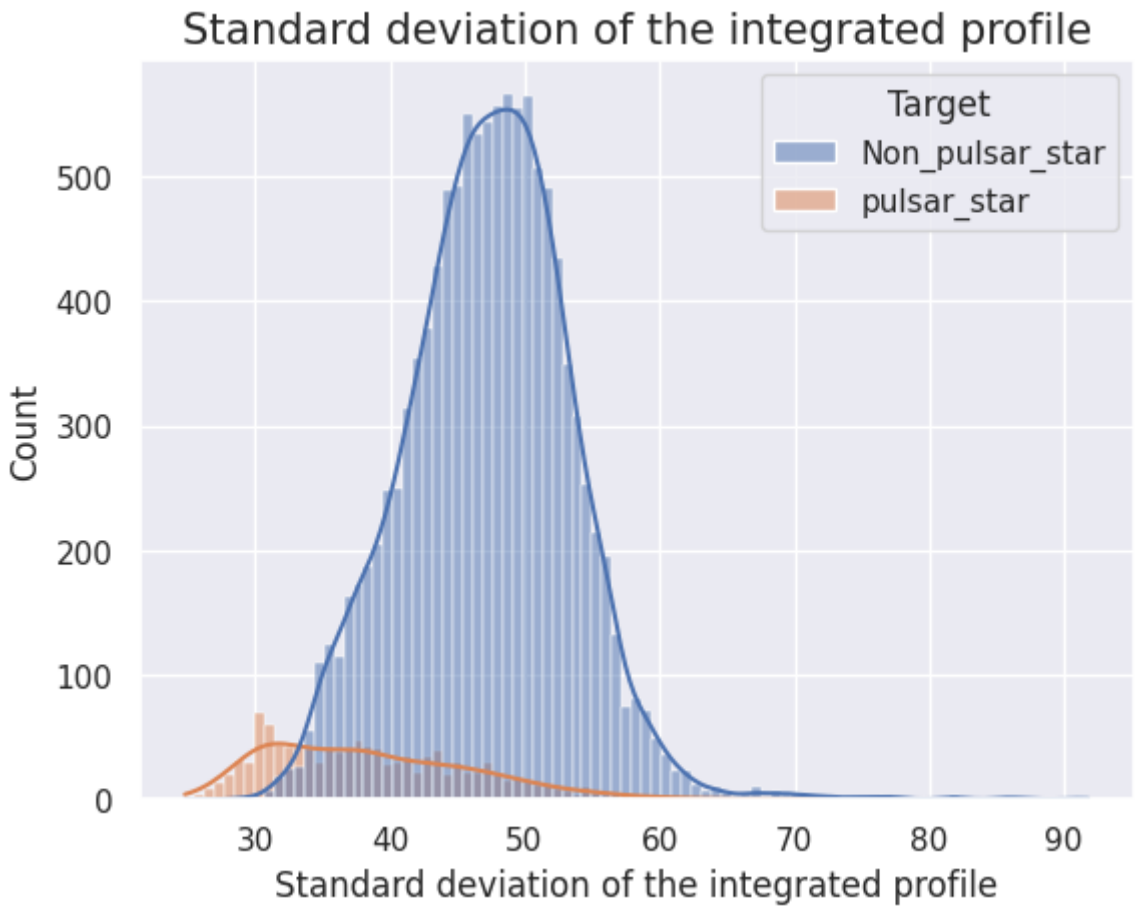
In [ ]:
```
Knn = KNNImputer()
X=Knn.fit_transform(Data_train)
```

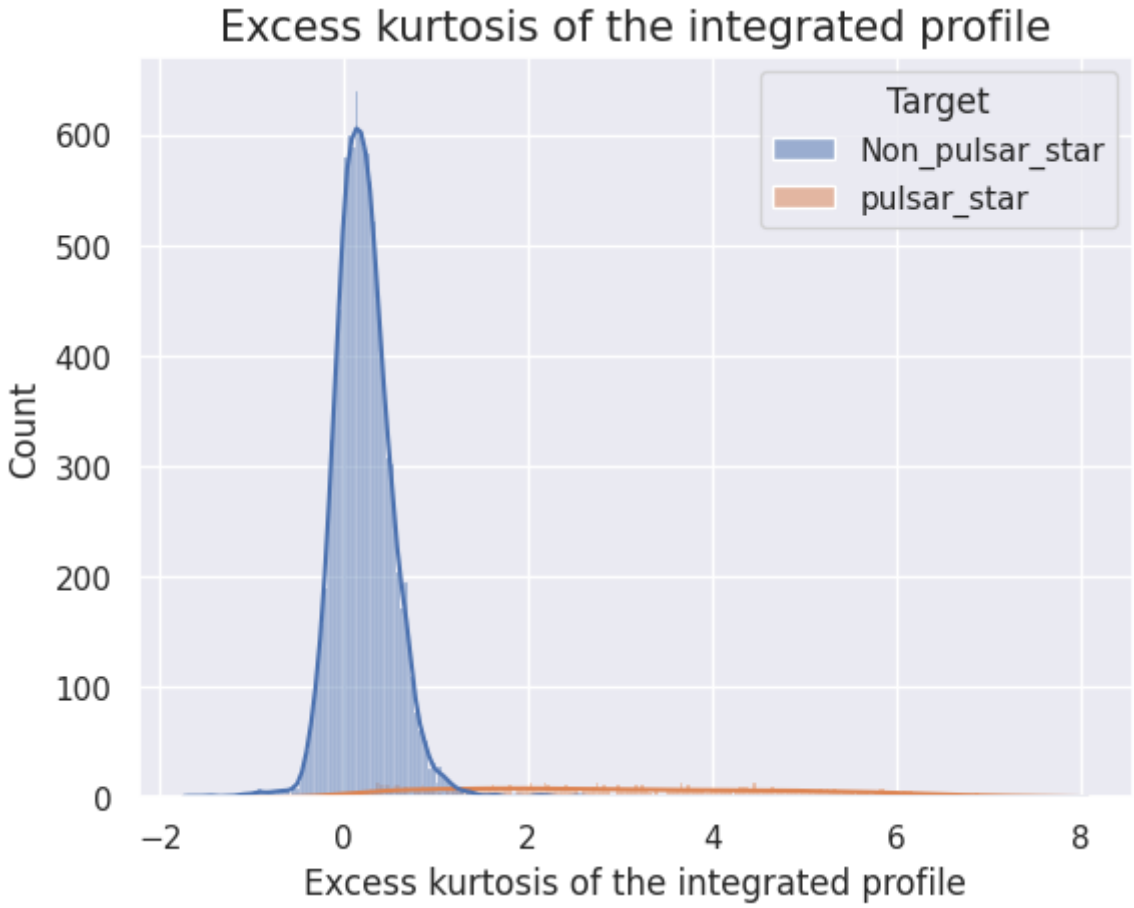
In [ ]:
```
New_data=pd.DataFrame(X,columns=Data_train.columns)
```

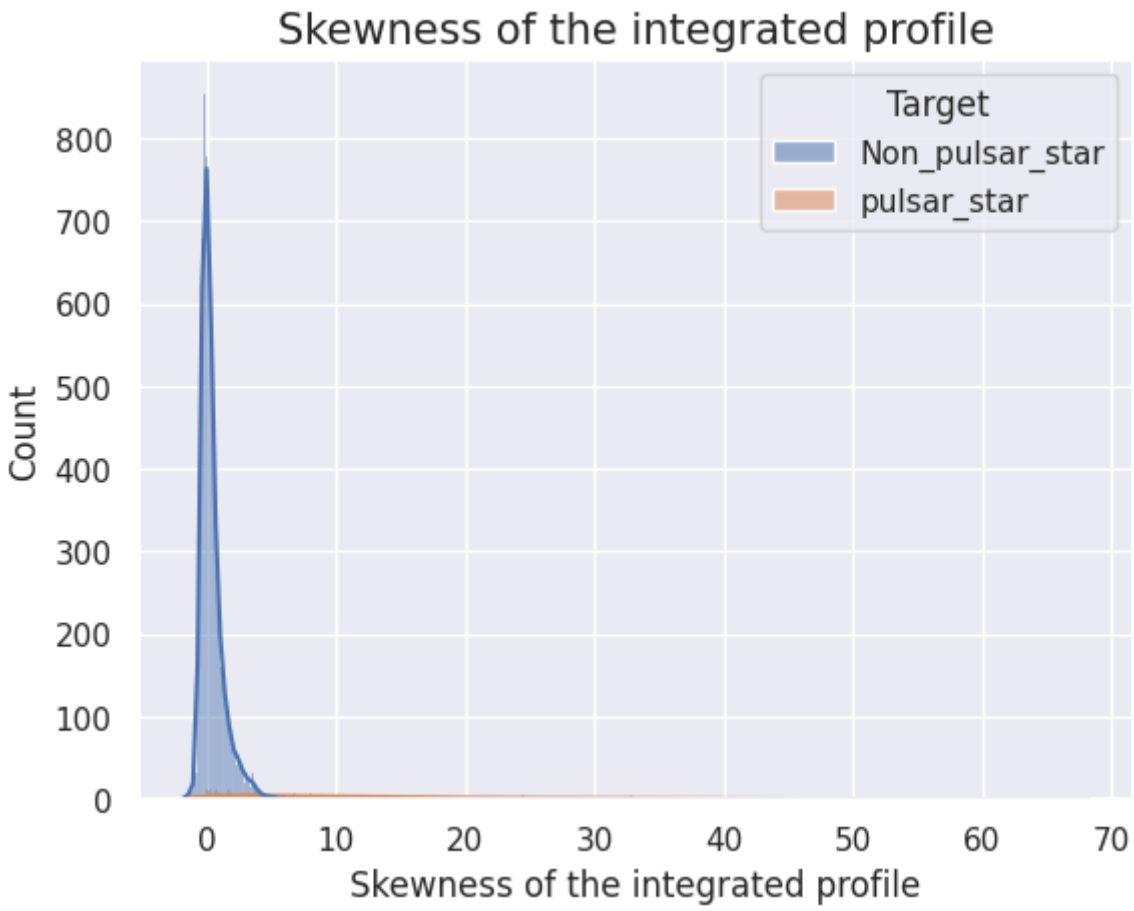
In [ ]:
```
New_data['Target']=Target
```

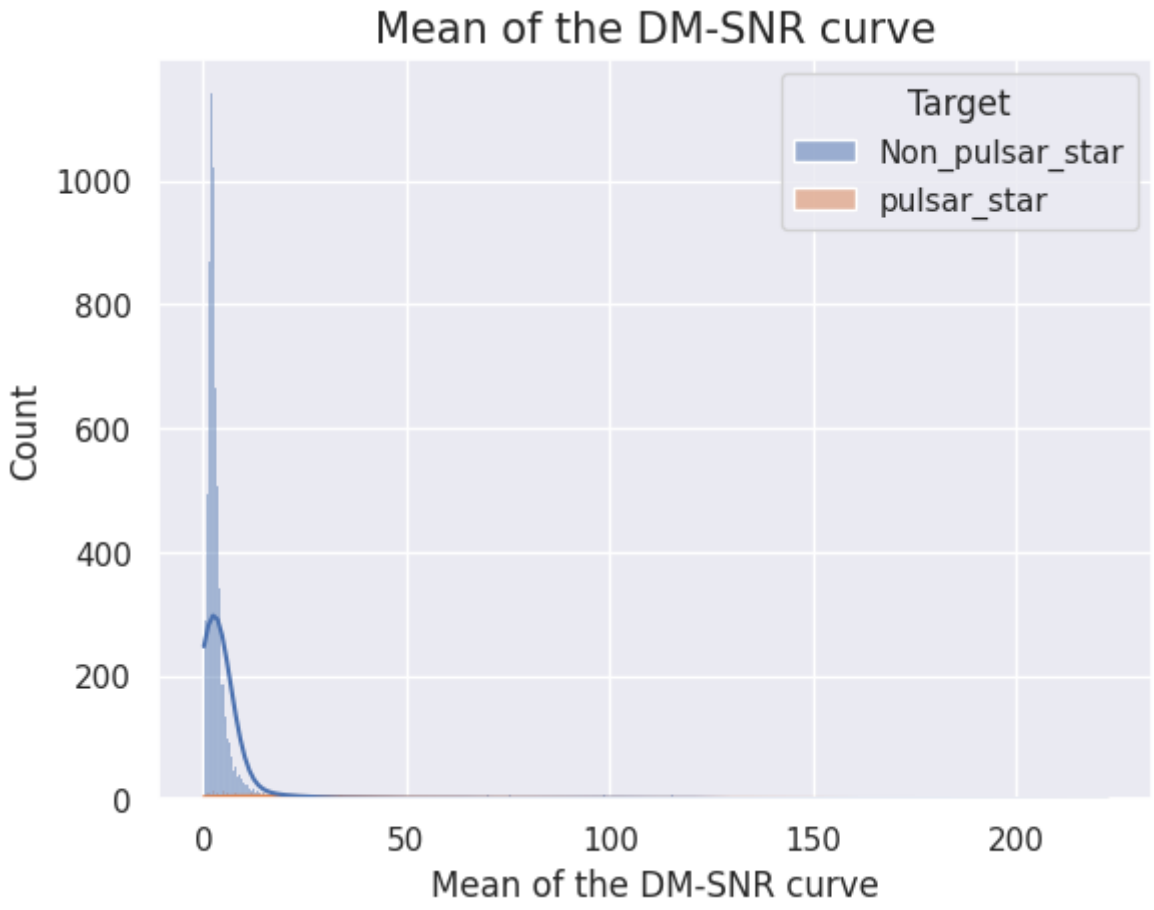
In [ ]:
```
Plot_dist(New_data)# Recheck the distribution after imputation
```

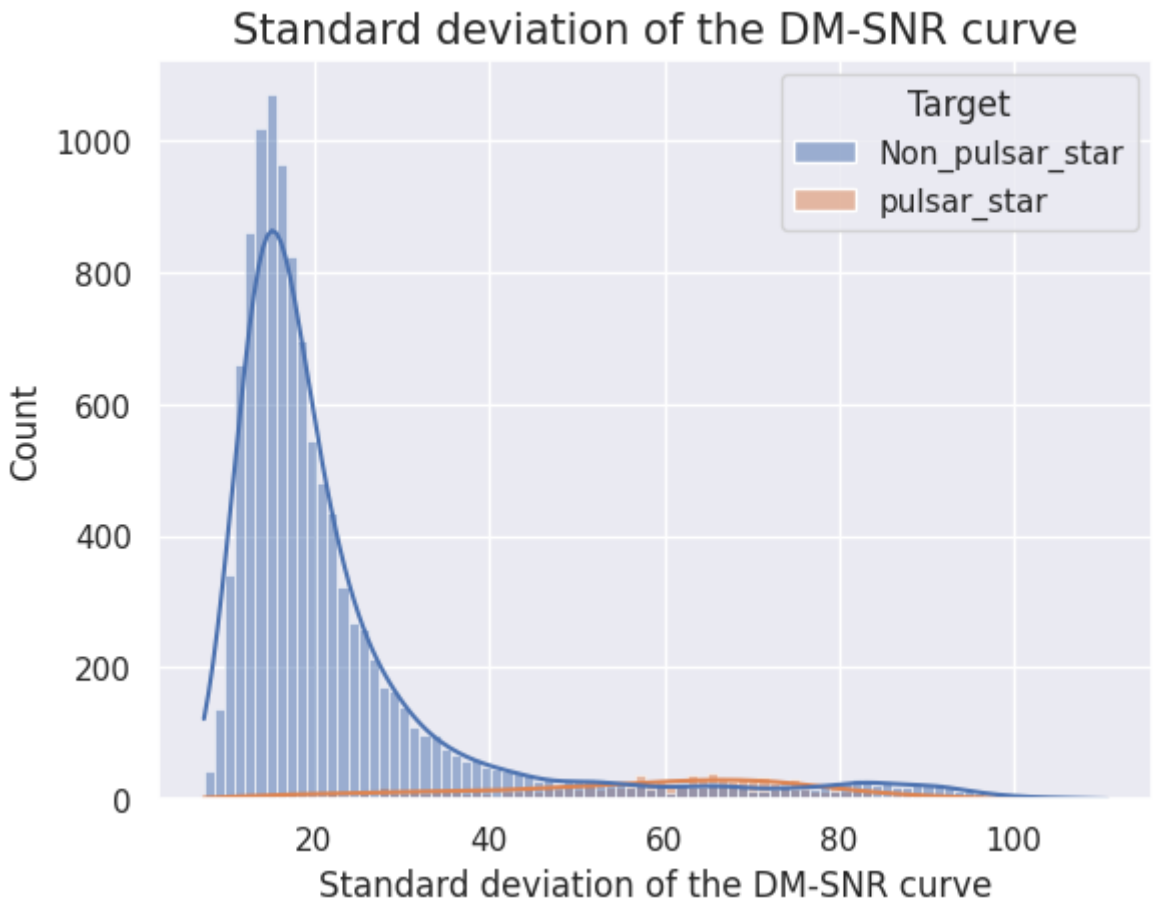

Mean of the integrated profile
Target
Non_pulsar_star
pulsar_star
Count
0
100
200
300
400
500
600
0
25
50
75
100
125
150
175
Mean of the integrated profile

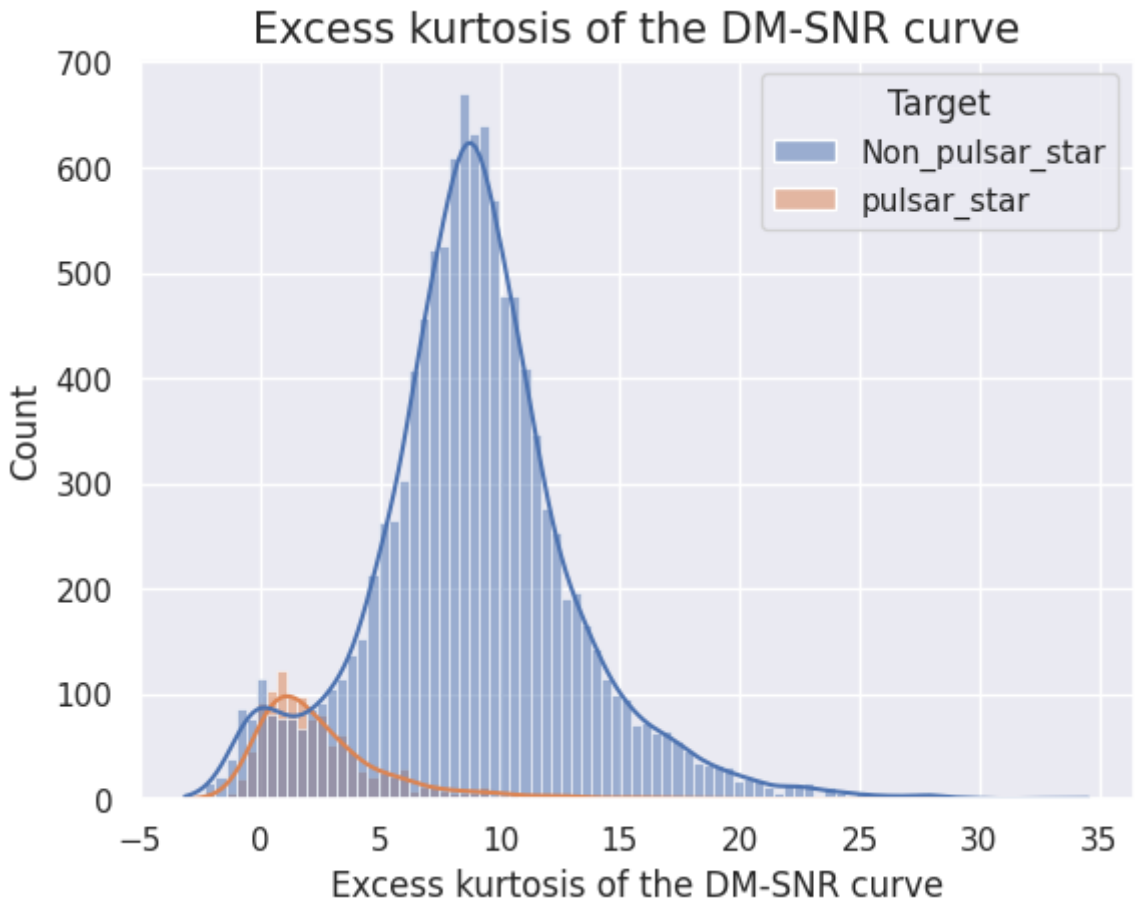

Standard deviation of the integrated profile
Target
Non_pulsar_star
pulsar_star
Count
0
100
200
300
400
500
30
40
50
60
70
80
90
Standard deviation of the integrated profile

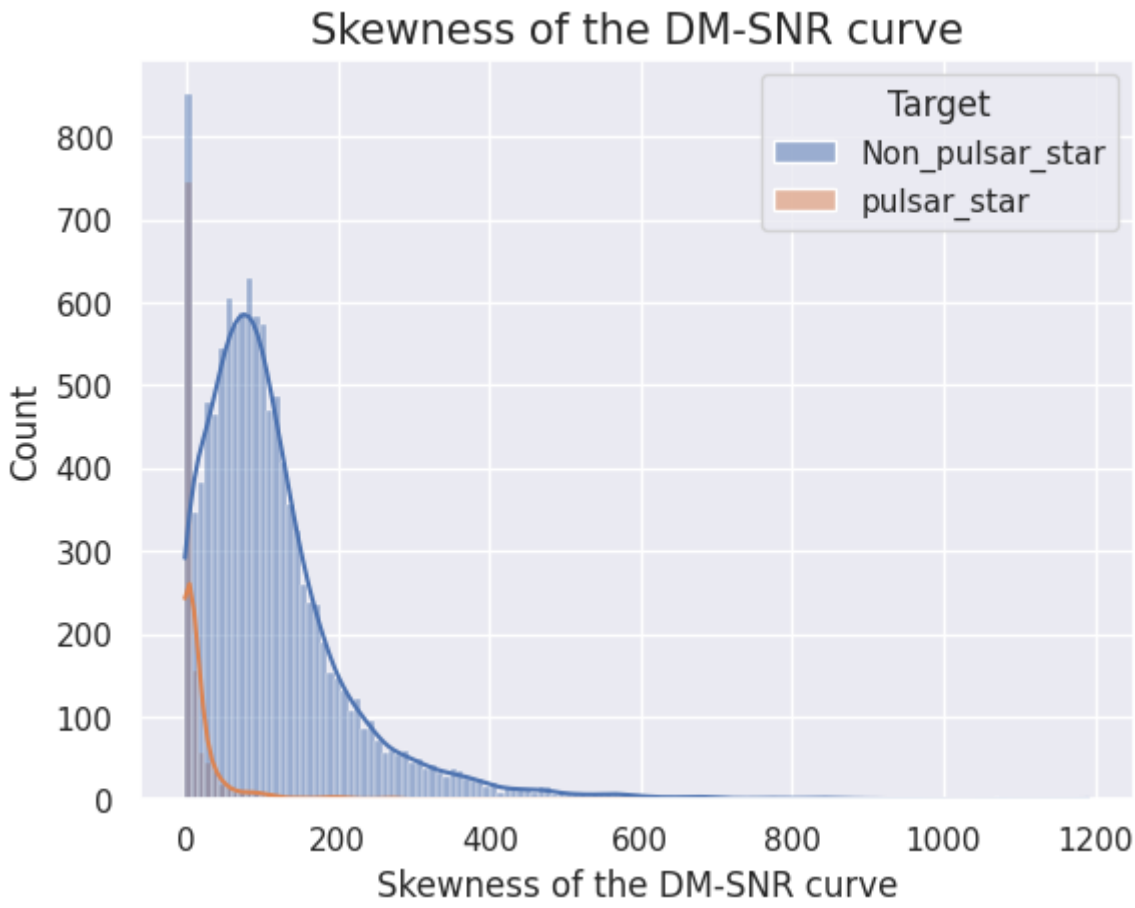

Excess kurtosis of the integrated profile
Target
Non_pulsar_star
pulsar_star
Count
600
500
400
300
200
100
0
−2
0
2
4
6
8
Excess kurtosis of the integrated profile


Skewness of the integrated profile
Target
Non_pulsar_star
pulsar_star
Count
800
700
600
500
400
300
200
100
0
0
10
20
30
40
50
60
70
Skewness of the integrated profile


Mean of the DM-SNR curve
Target
Non_pulsar_star
pulsar_star
Count
0
200
400
600
800
1000
0
50
100
150
200
Mean of the DM-SNR curve


Standard deviation of the DM-SNR curve
Target
Non_pulsar_star
pulsar_star
Count
0
200
400
600
800
1000
20
40
60
80
100
Standard deviation of the DM-SNR curve

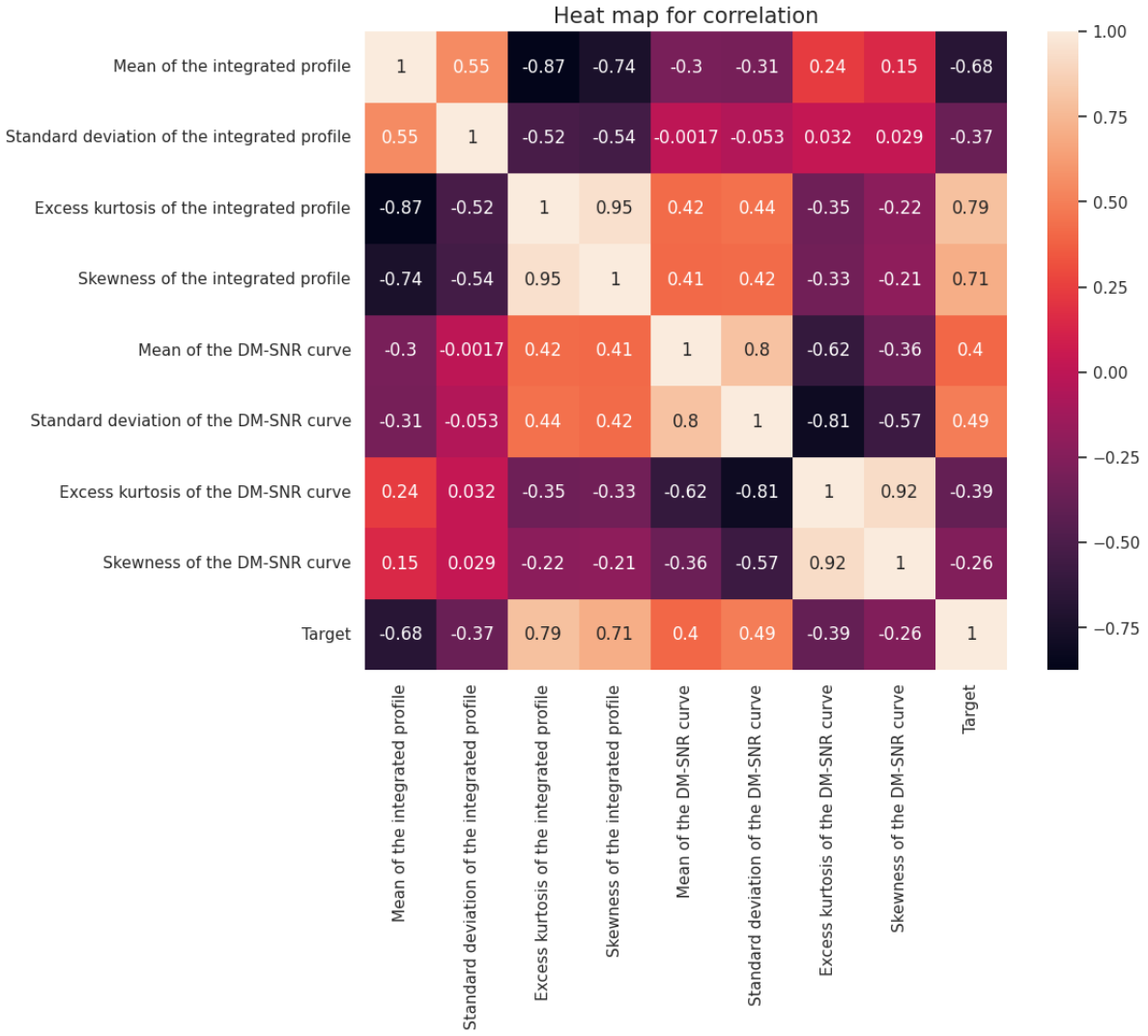

Excess kurtosis of the DM-SNR curve
Target
Non_pulsar_star
pulsar_star
Count
700
600
500
400
300
200
100
0
−5
0
5
10
15
20
25
30
35
Excess kurtosis of the DM-SNR curve


Skewness of the DM-SNR curve
Target
Non_pulsar_star
pulsar_star
Count
800
700
600
500
400
300
200
100
0
0
200
400
600
800
1000
1200
Skewness of the DM-SNR curve

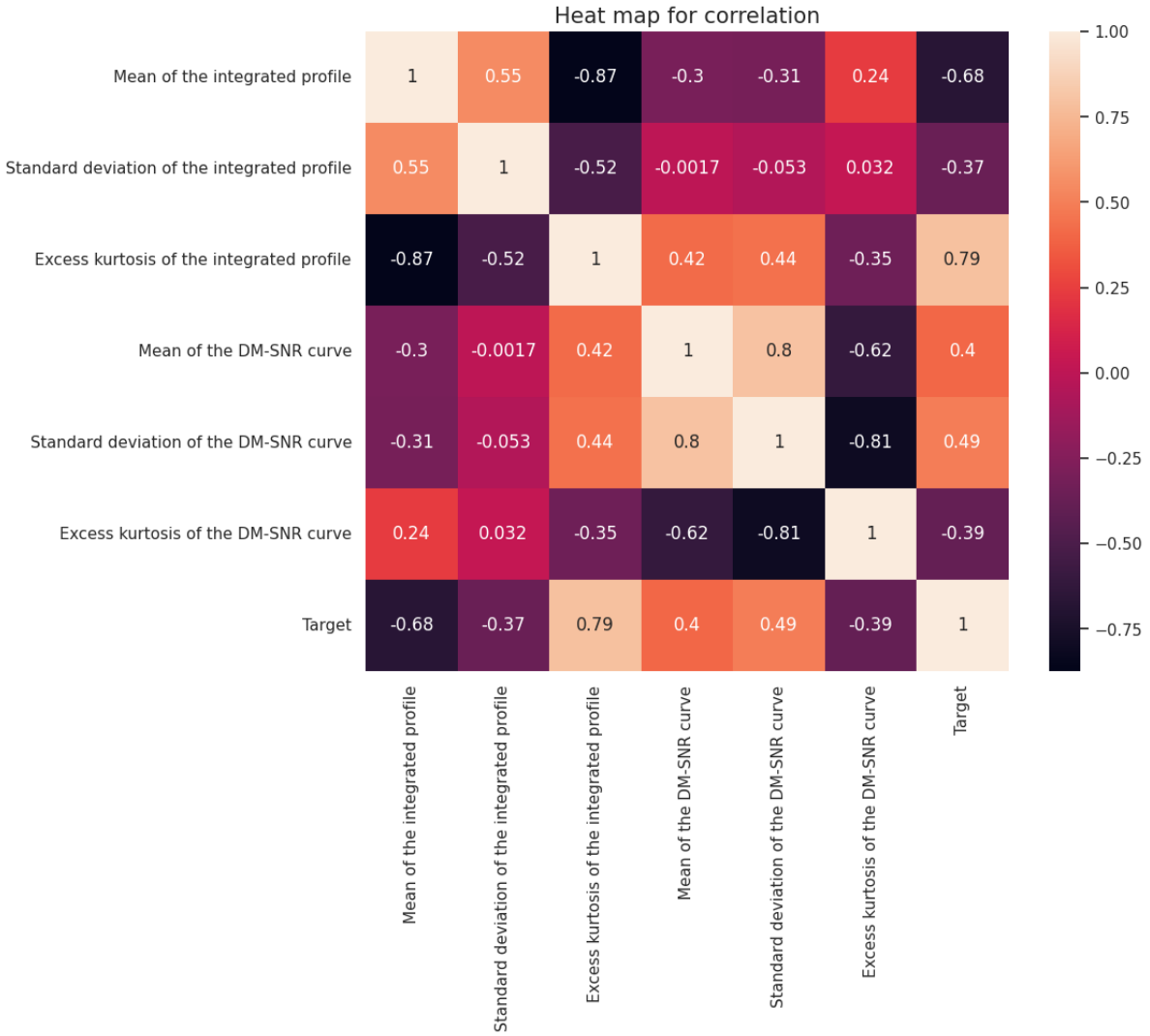
In [ ]:
```
New_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12528 entries, 0 to 12527
Data columns (total 9 columns):
 #   Column                                        Non-Null Count  Dtype  
---  ------                                        --------------  -----  
 0   Mean of the integrated profile                12528 non-null  float64
 1   Standard deviation of the integrated profile  12528 non-null  float64
 2   Excess kurtosis of the integrated profile     12528 non-null  float64
 3   Skewness of the integrated profile            12528 non-null  float64
 4   Mean of the DM-SNR curve                      12528 non-null  float64
 5   Standard deviation of the DM-SNR curve        12528 non-null  float64
 6   Excess kurtosis of the DM-SNR curve           12528 non-null  float64
 7   Skewness of the DM-SNR curve                  12528 non-null  float64
 8   Target                                        12528 non-null  int64  
dtypes: float64(8), int64(1)
memory usage: 881.0 KB
```

In [ ]:
```
Correlation_plot(New_data)
```



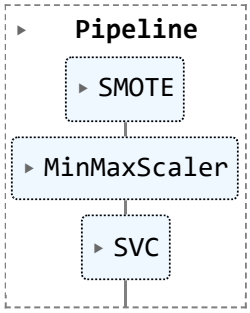
In [ ]:
```
#Drop the features having correlation coefficient >0.9
cols_to_drop=['Skewness of the integrated profile','Skewness of the DM-SNR curve']
New_data=New_data.drop(cols_to_drop,axis=1)
Correlation_plot(New_data)
```



Training the data

In [ ]:
```
Y=New_data.pop('Target')
clf=SVC(kernel='poly',C=0.1,degree=10)
bag=BaggingClassifier(estimator=clf,n_estimators=100,bootstrap_features=True,max_fe
pipe=Pipeline([('oversample',SMOTE()),('minmax',MinMaxScaler()),('classifier',clf)]
pipe.fit(New_data,Y)
```

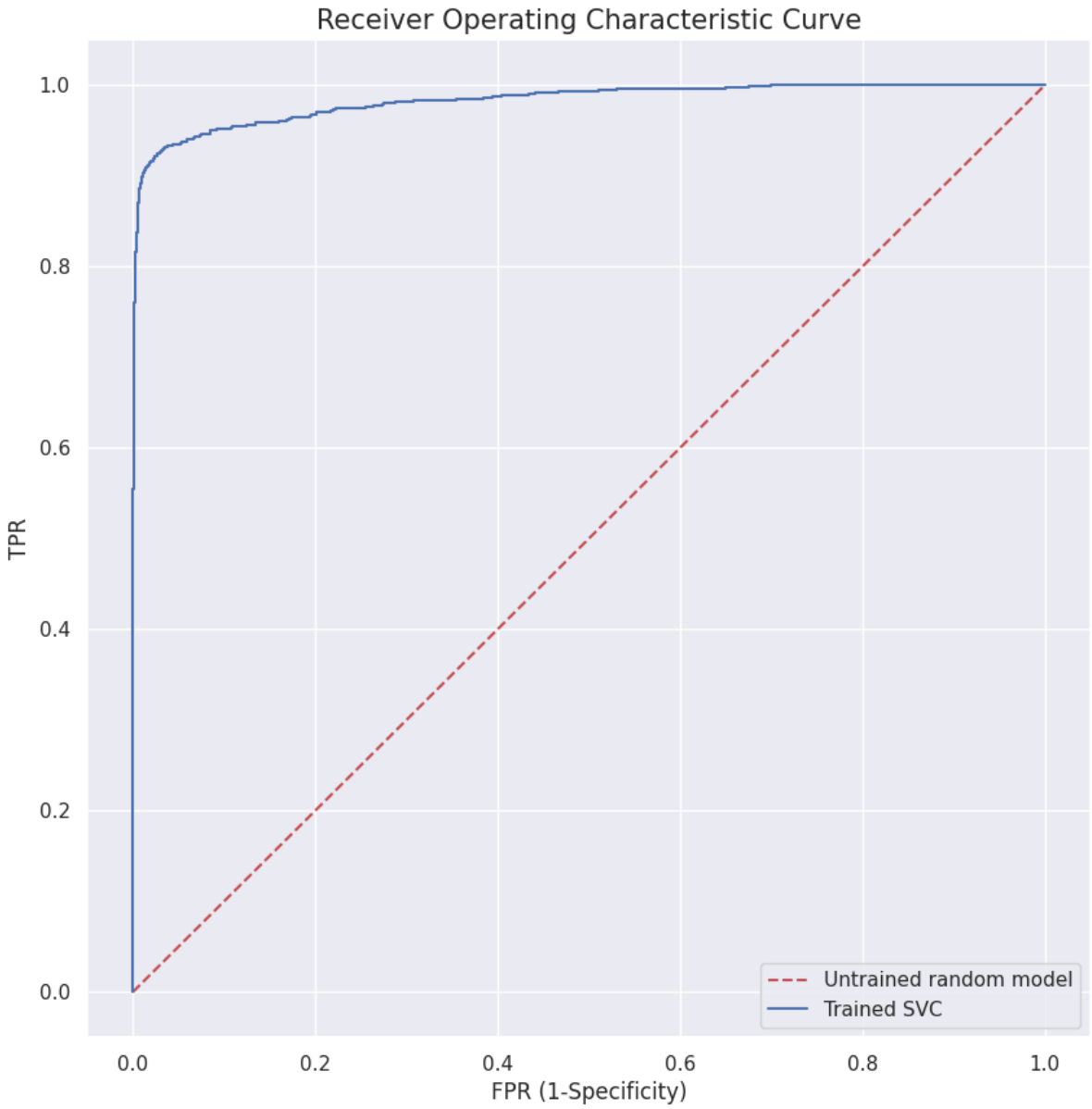
Out[ ]:



In [ ]:
```
print(classification_report(Y,pipe.predict(New_data)))
```

	precision	recall	f1-score	support
0	0.99	0.98	0.99	11375
1	0.85	0.91	0.88	1153
accuracy			0.98	12528
macro avg	0.92	0.95	0.93	12528
weighted avg	0.98	0.98	0.98	12528

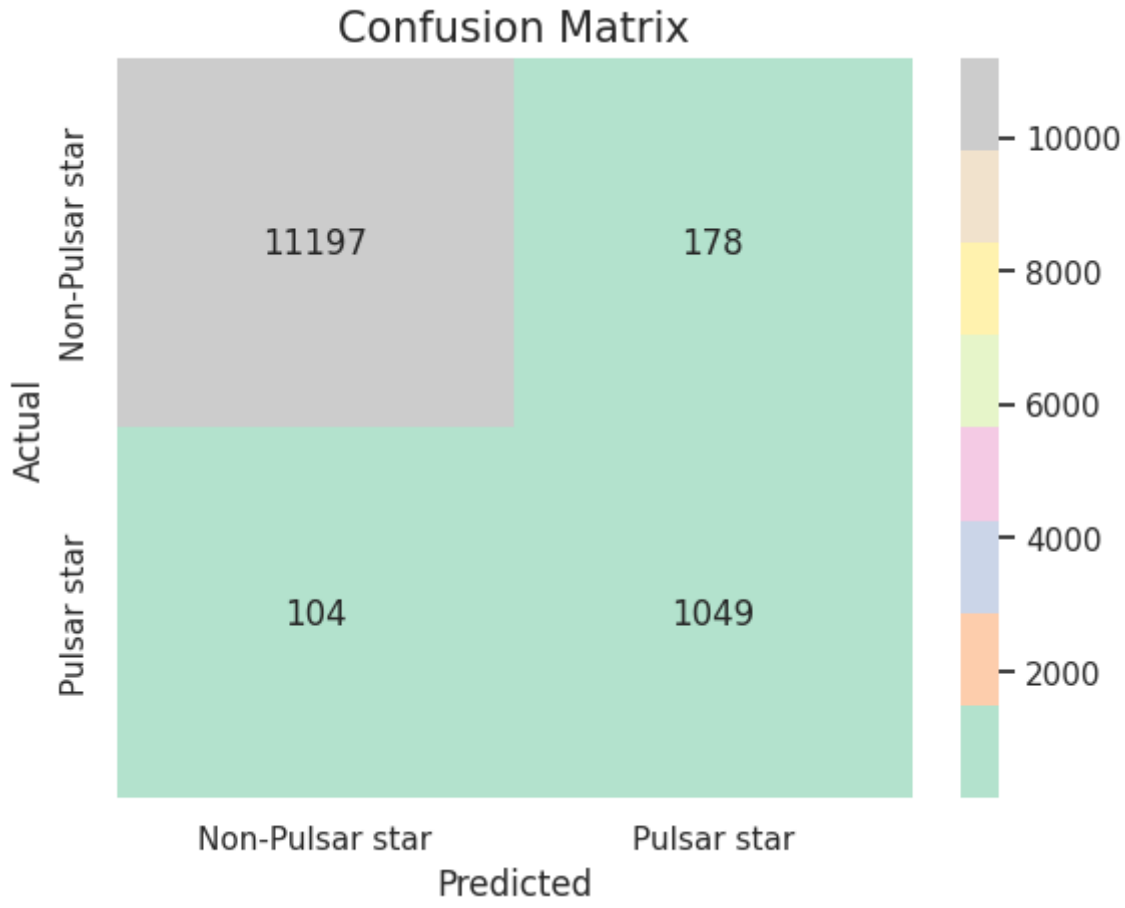
In [ ]:
```
Y_test_proba=pipe.decision_function(New_data)
fpr, tpr, thresholds = roc_curve(Y, Y_test_proba)
sns.set()
plt.figure(figsize=(10,10))
plt.plot([0,1],[0,1],'r--',label='Untrained random model')
plt.plot(fpr,tpr,label='Trained SVC')
plt.title("Receiver Operating Characteristic Curve",fontsize=15)
plt.ylabel("TPR")
plt.xlabel("FPR (1-Specificity)")
plt.legend()
plt.show()
```



In [ ]:
```
auc(fpr,tpr)# AUC score
```

Out[ ]:
```
0.9826389256883619
```

In [ ]:
```
cm=confusion_matrix(Y,pipe.predict(New_data))
plt.title("Confusion Matrix",fontsize=15)
sns.heatmap(cm,annot=True,cmap='Pastel2',fmt='d')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.xticks(ticks=[0.5,1.5],labels=['Non-Pulsar star','Pulsar star'])
plt.yticks(ticks=[0.5,1.5],labels=['Non-Pulsar star','Pulsar star'])
plt.show()
```



Load the test set

In [ ]:
```
Data_test=pd.read_excel(r'/content/drive/MyDrive/DAL dataset/Assignment 6/pulsar_da
Data_test.columns = Data_test.columns.str.strip()
```

In [ ]:
```
Data_test.info()
```

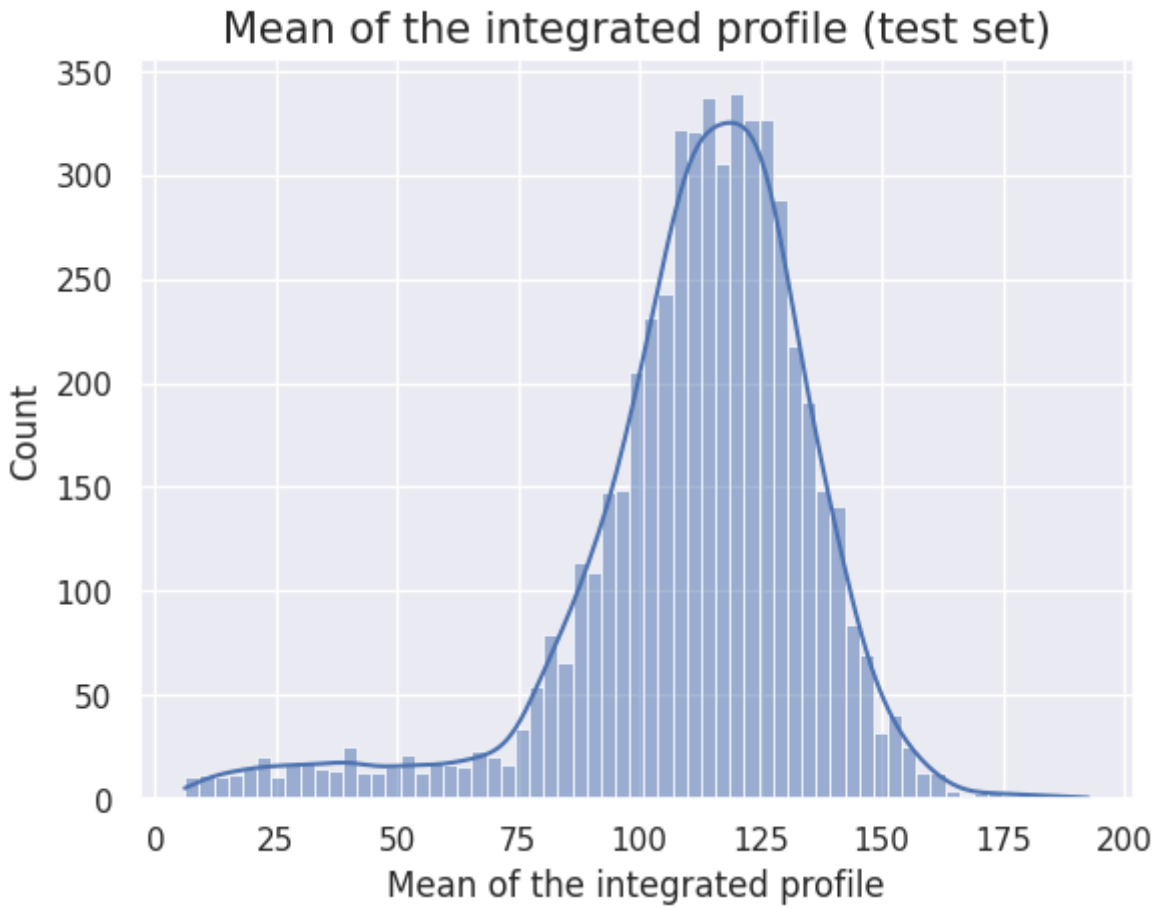
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5370 entries, 0 to 5369
Data columns (total 9 columns):
 #   Column                                        Non-Null Count  Dtype
---  ------                                        --------------  -----
 0   Mean of the integrated profile                5370 non-null   float64
 1   Standard deviation of the integrated profile  5370 non-null   float64
 2   Excess kurtosis of the integrated profile     4603 non-null   float64
 3   Skewness of the integrated profile            5370 non-null   float64
 4   Mean of the DM-SNR curve                      5370 non-null   float64
 5   Standard deviation of the DM-SNR curve        4846 non-null   float64
 6   Excess kurtosis of the DM-SNR curve           5370 non-null   float64
 7   Skewness of the DM-SNR curve                  5126 non-null   float64
 8   target_class                                  0 non-null      float64
dtypes: float64(9)
memory usage: 377.7 KB
```

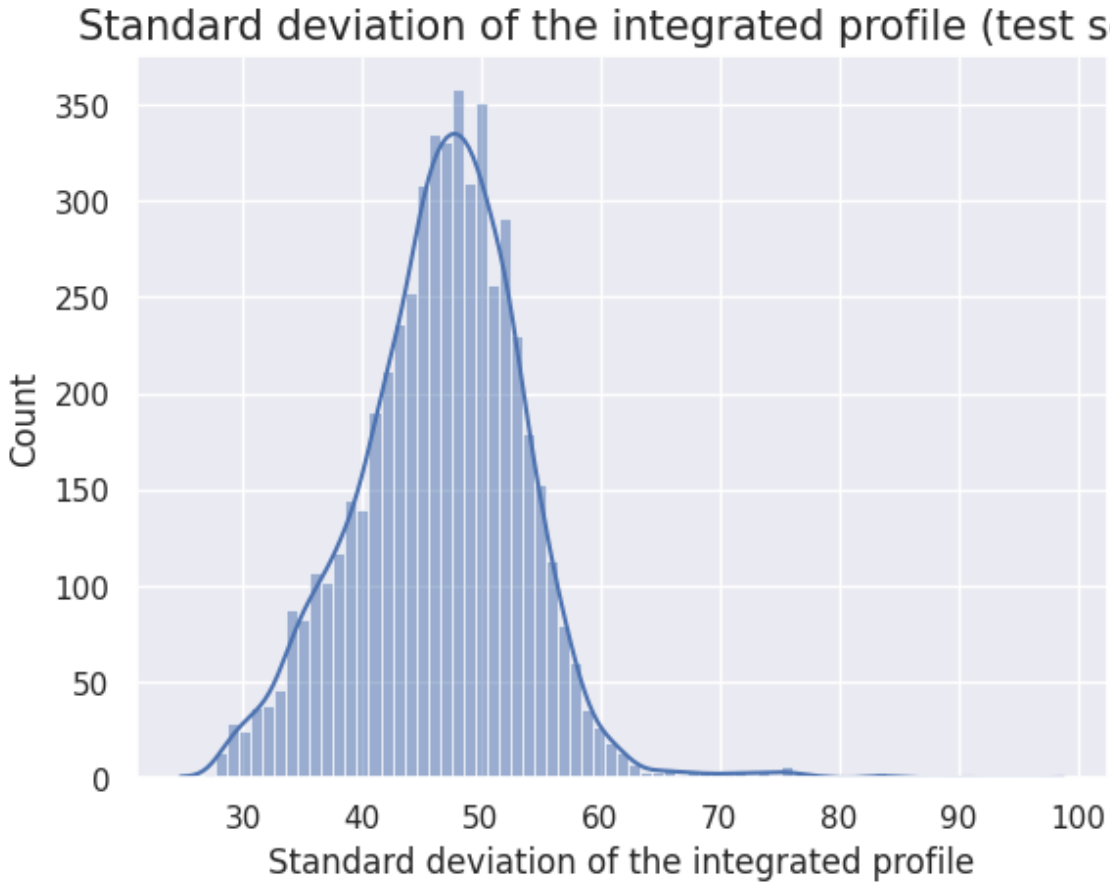
In [ ]:
```
Data_test=Data_test.drop(['target_class'],axis=1)
```

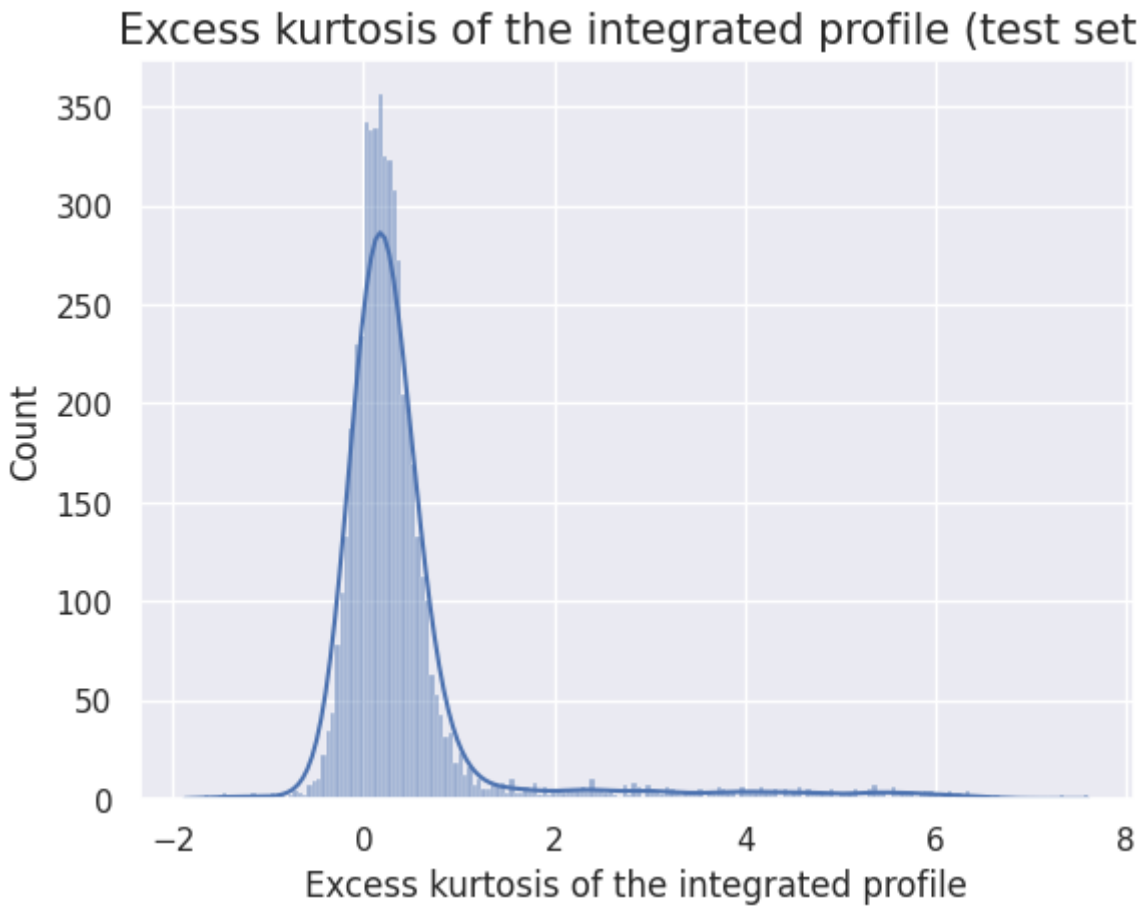
In [ ]:
```
X_test=Knn.transform(Data_test)
New_data_test=pd.DataFrame(X_test,columns=Data_test.columns)
```

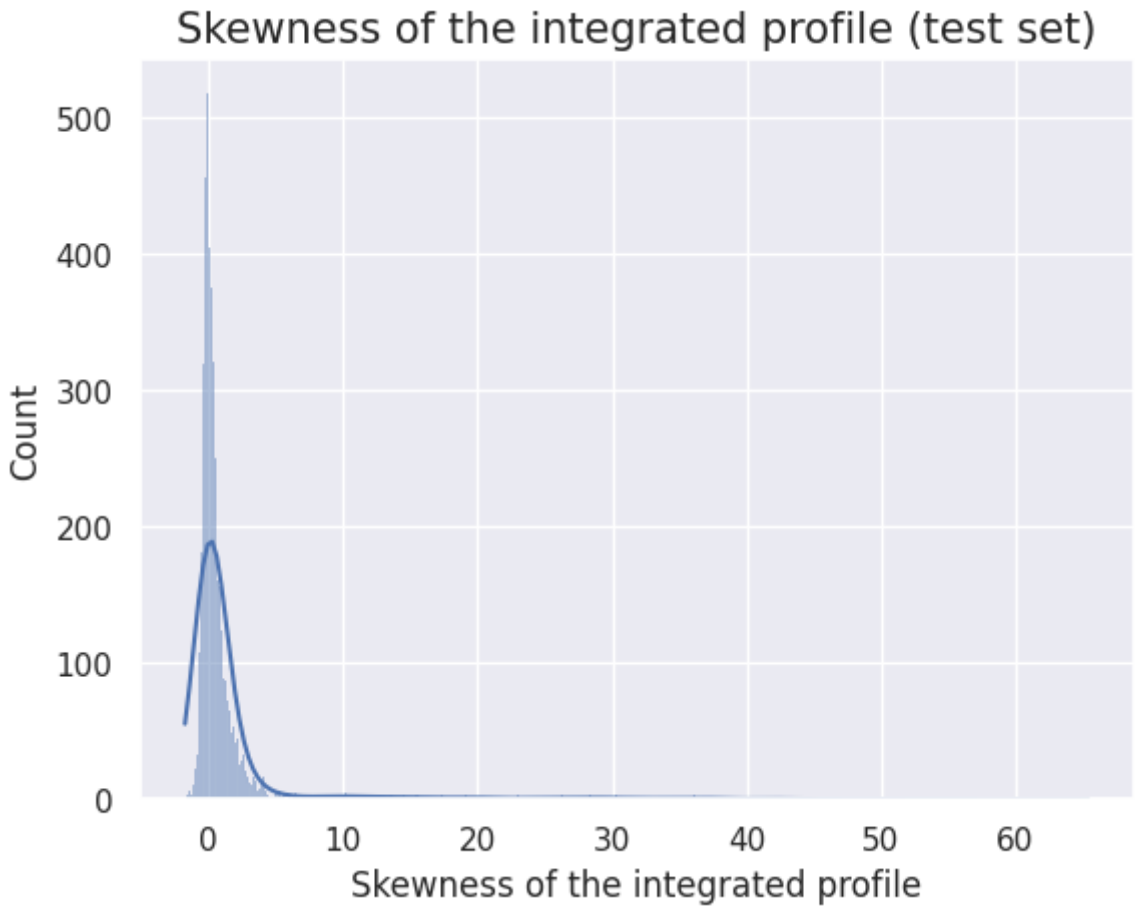
Visualization on the test set

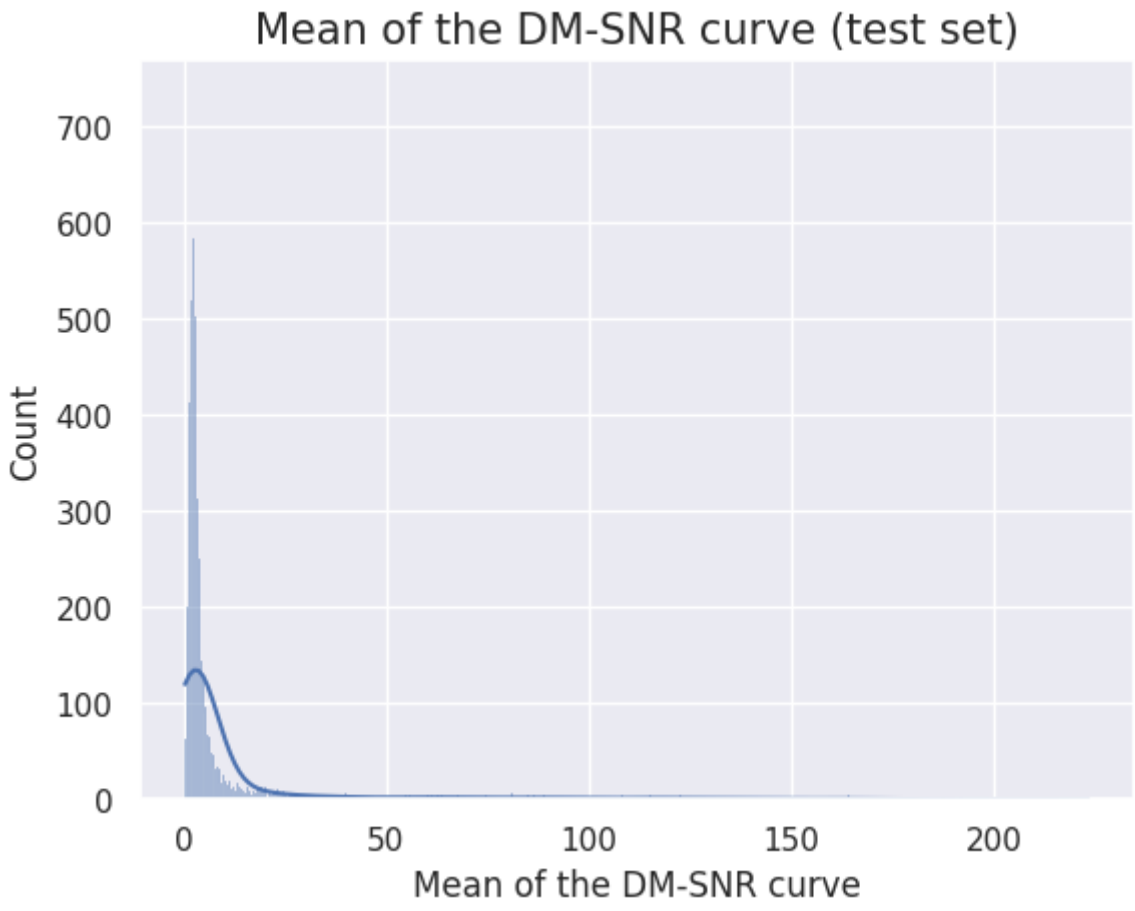
In [ ]:
```
Columns=New_data_test.columns
for col in Columns:
  plt.title(col+' (test set)',fontsize=15)
  sns.histplot(New_data_test,x=col,kde=True)
  plt.show()
```

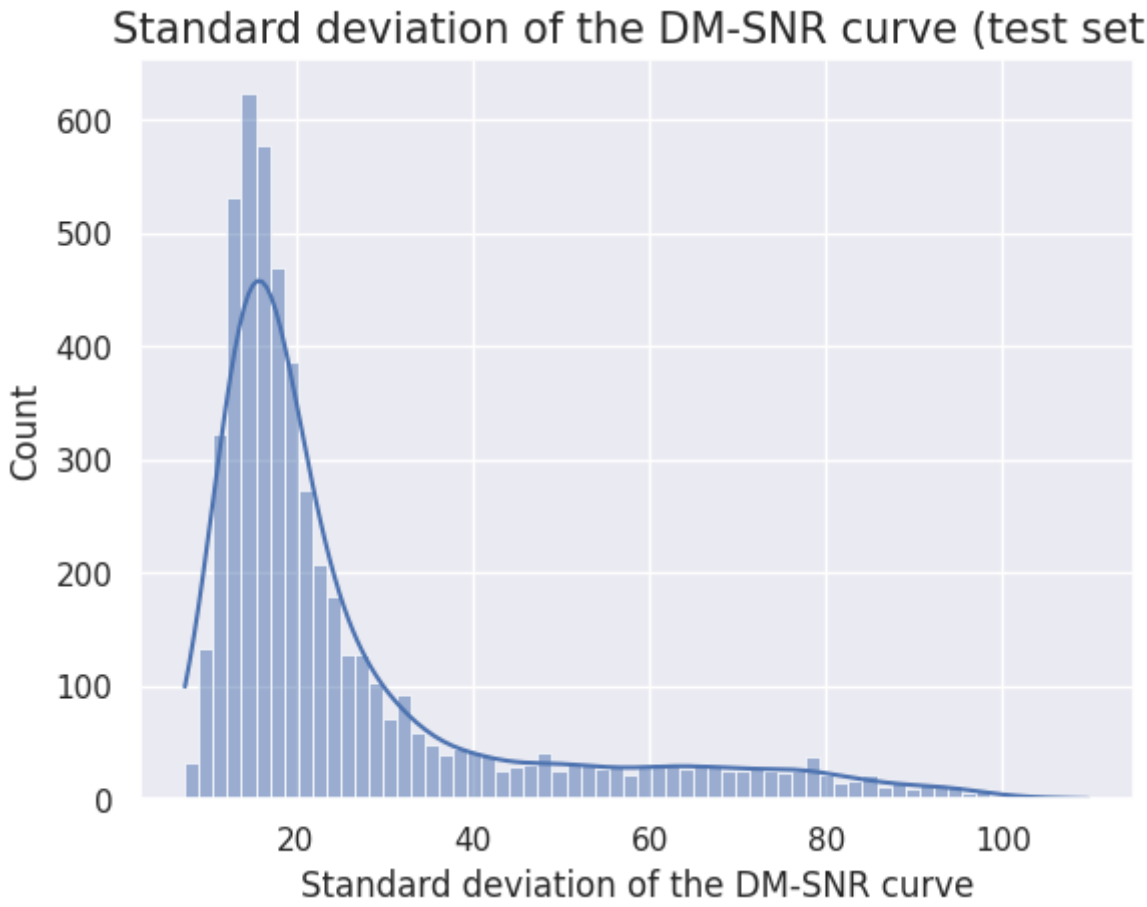


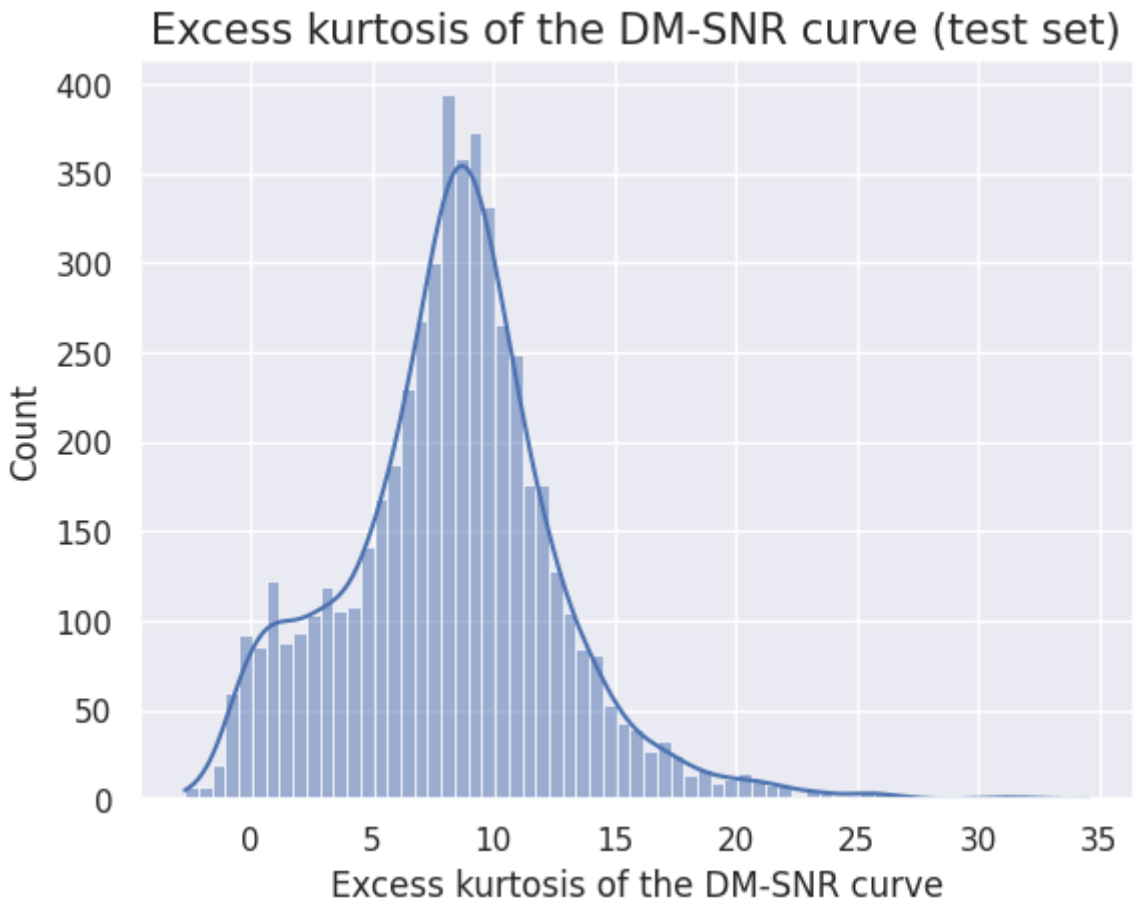

Standard deviation of the integrated profile (test set)
Count
0
50
100
150
200
250
300
350
30
40
50
60
70
80
90
100
Standard deviation of the integrated profile

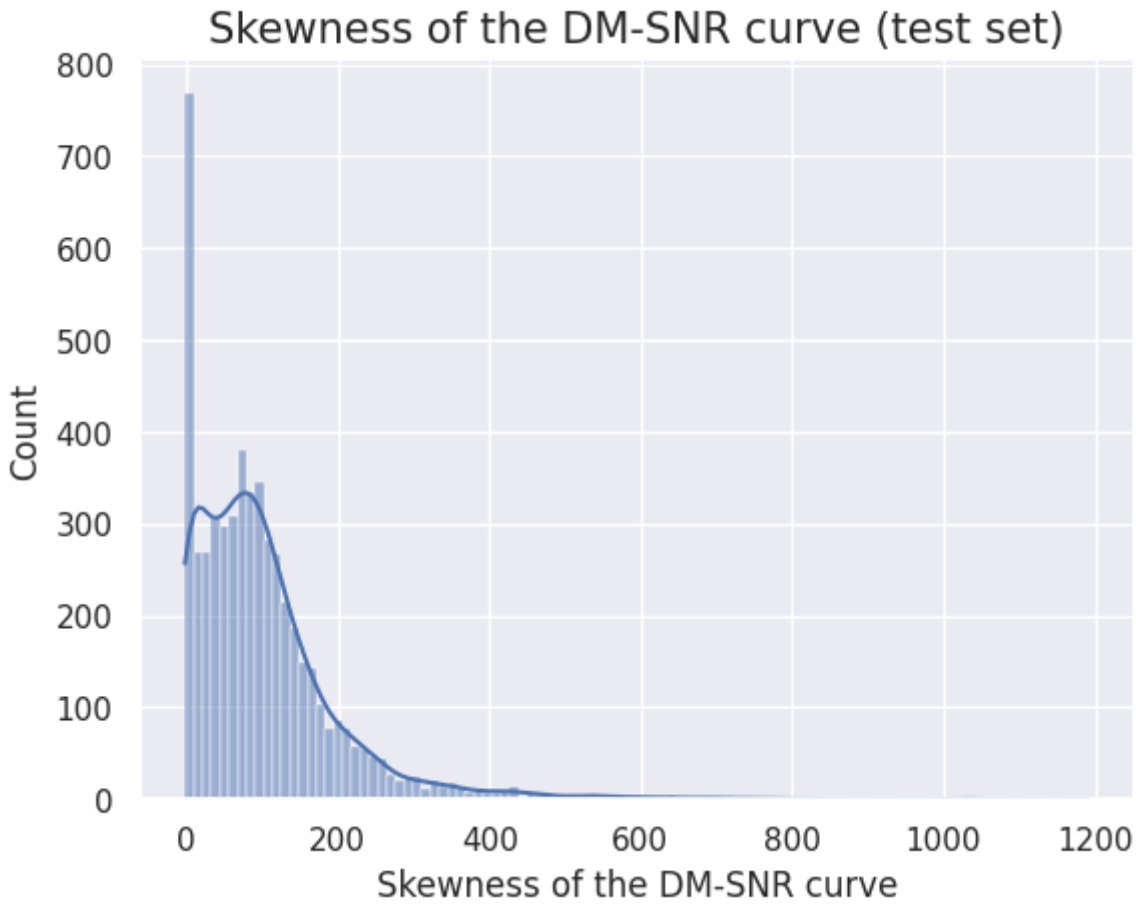

Excess kurtosis of the integrated profile (test set)
Count
0
50
100
150
200
250
300
350
−2
0
2
4
6
8
Excess kurtosis of the integrated profile


Skewness of the integrated profile (test set)
Count
0
100
200
300
400
500
0
10
20
30
40
50
60
Skewness of the integrated profile


Mean of the DM-SNR curve (test set)
Count
0
100
200
300
400
500
600
700
0
50
100
150
200
Mean of the DM-SNR curve


Standard deviation of the DM-SNR curve (test set)
Count
0
100
200
300
400
500
600
20
40
60
80
100
Standard deviation of the DM-SNR curve


Excess kurtosis of the DM-SNR curve (test set)
Count
0
50
100
150
200
250
300
350
400
0
5
10
15
20
25
30
35
Excess kurtosis of the DM-SNR curve



In [ ]:
```
# Drop the correlated features
New_data_test=New_data_test.drop(cols_to_drop,axis=1)
Correlation_plot(New_data_test)
```

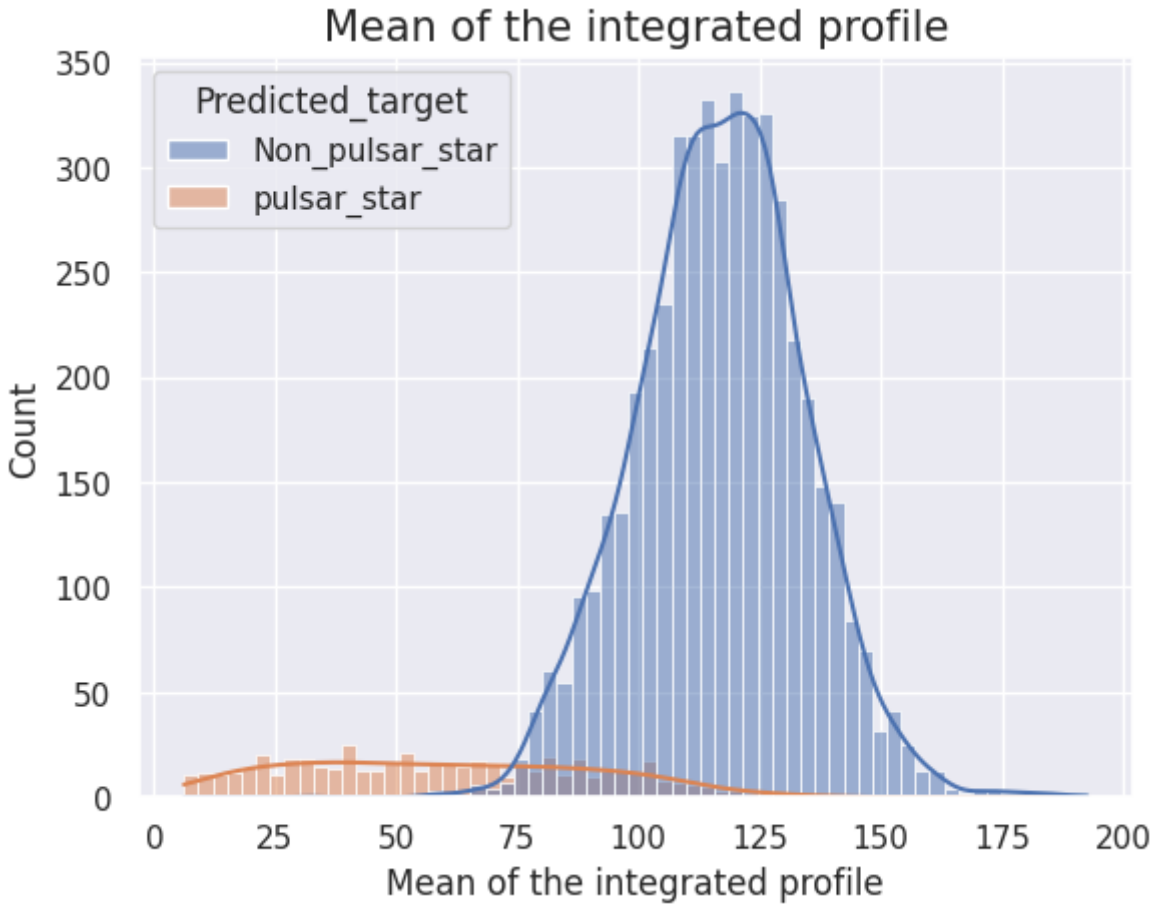
In [ ]:
```
Yhat_test=pipe.predict(New_data_test)
```

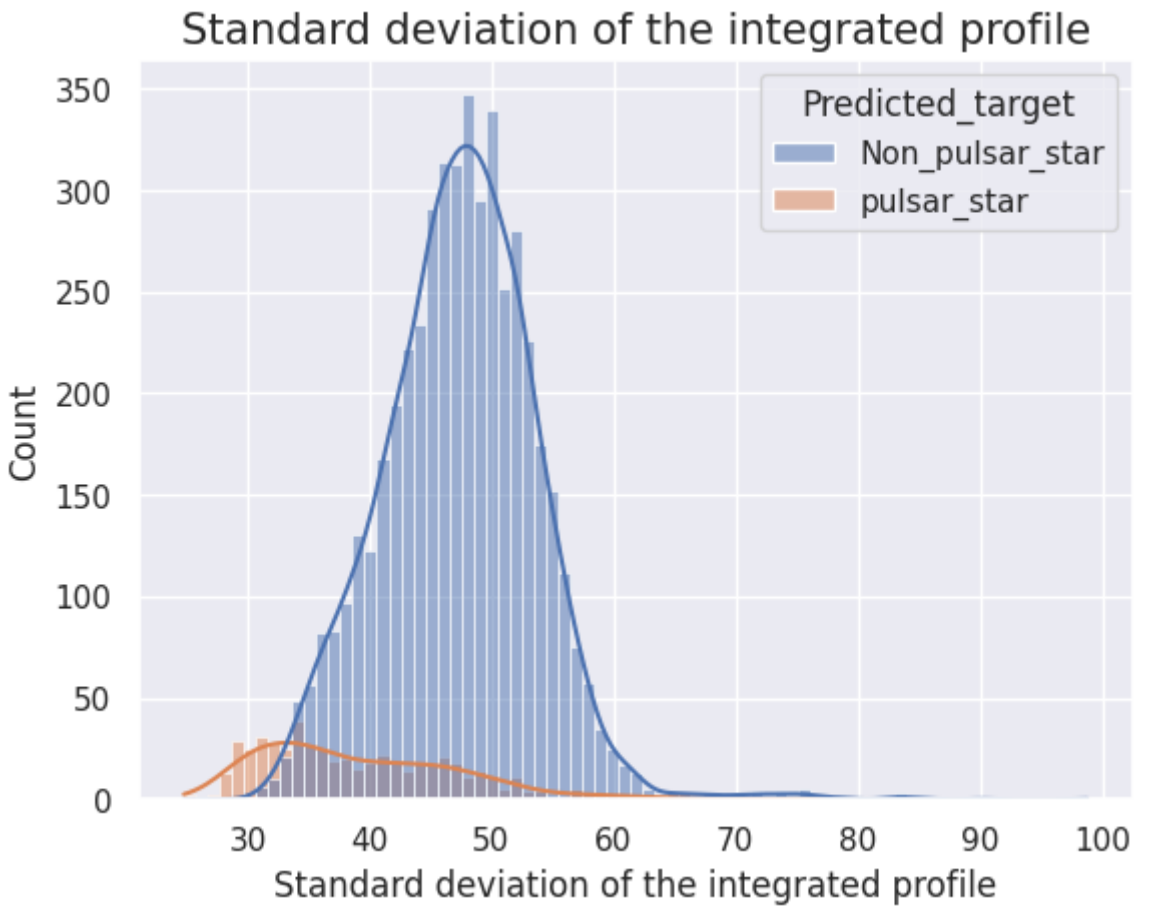
In [ ]:
```
np.unique(Yhat_test,return_counts=True)
```

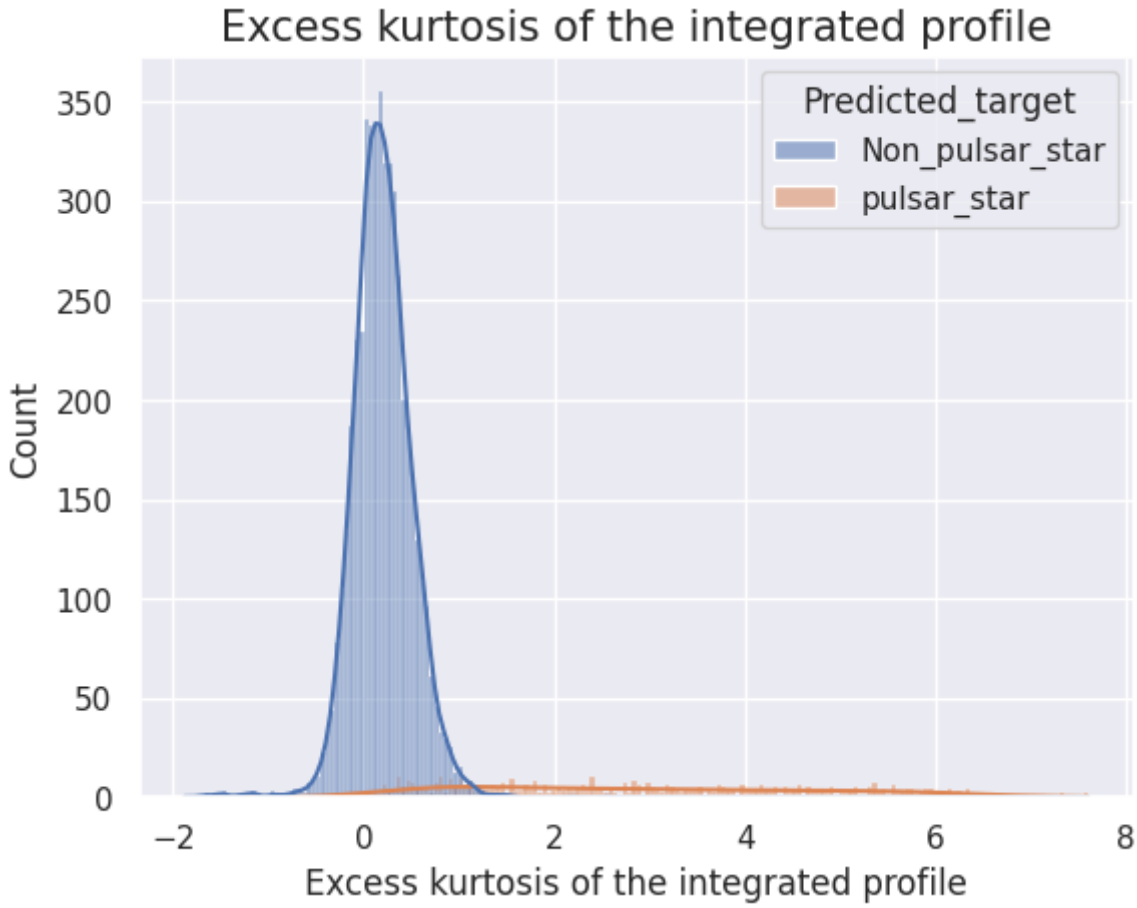
Out[ ]:
```
(array([0, 1]), array([4840,  530]))
```

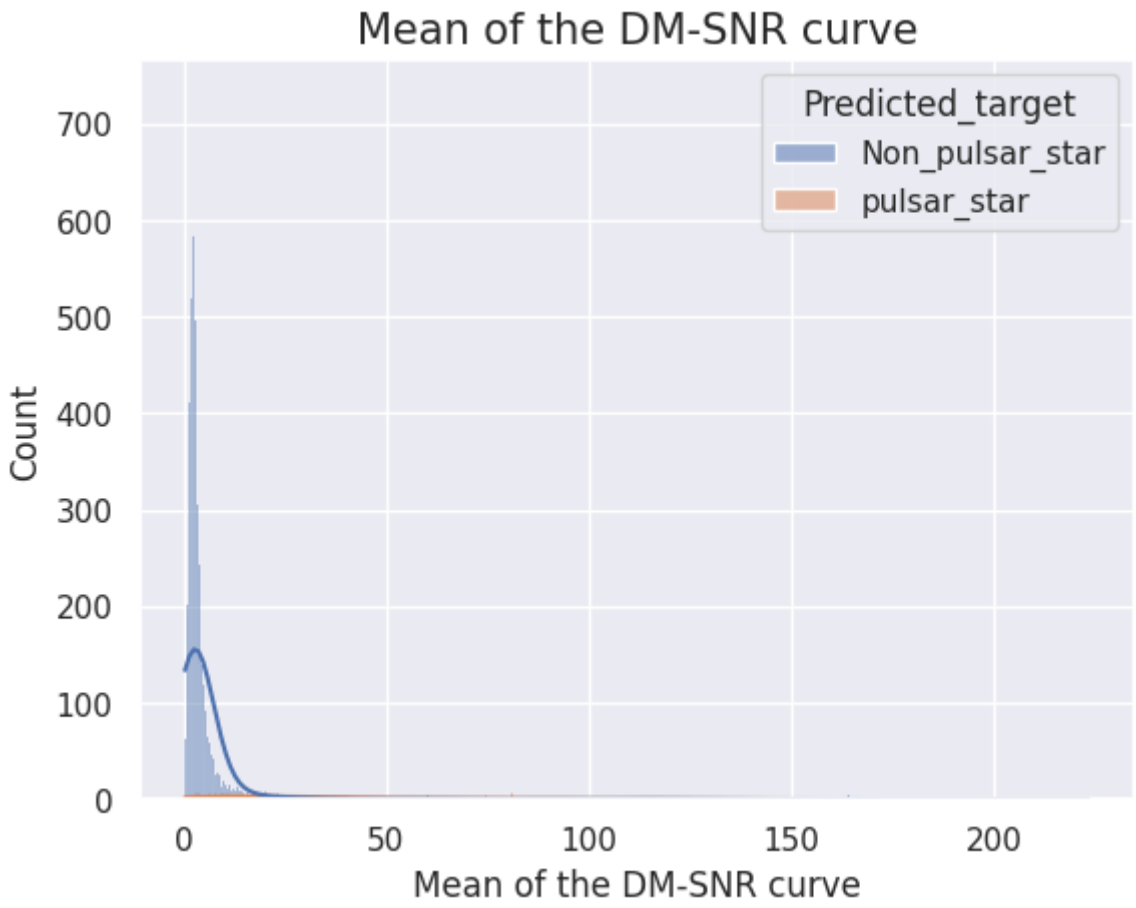
In [ ]:
```
New_data_test['Predicted_target']=Yhat_test
```

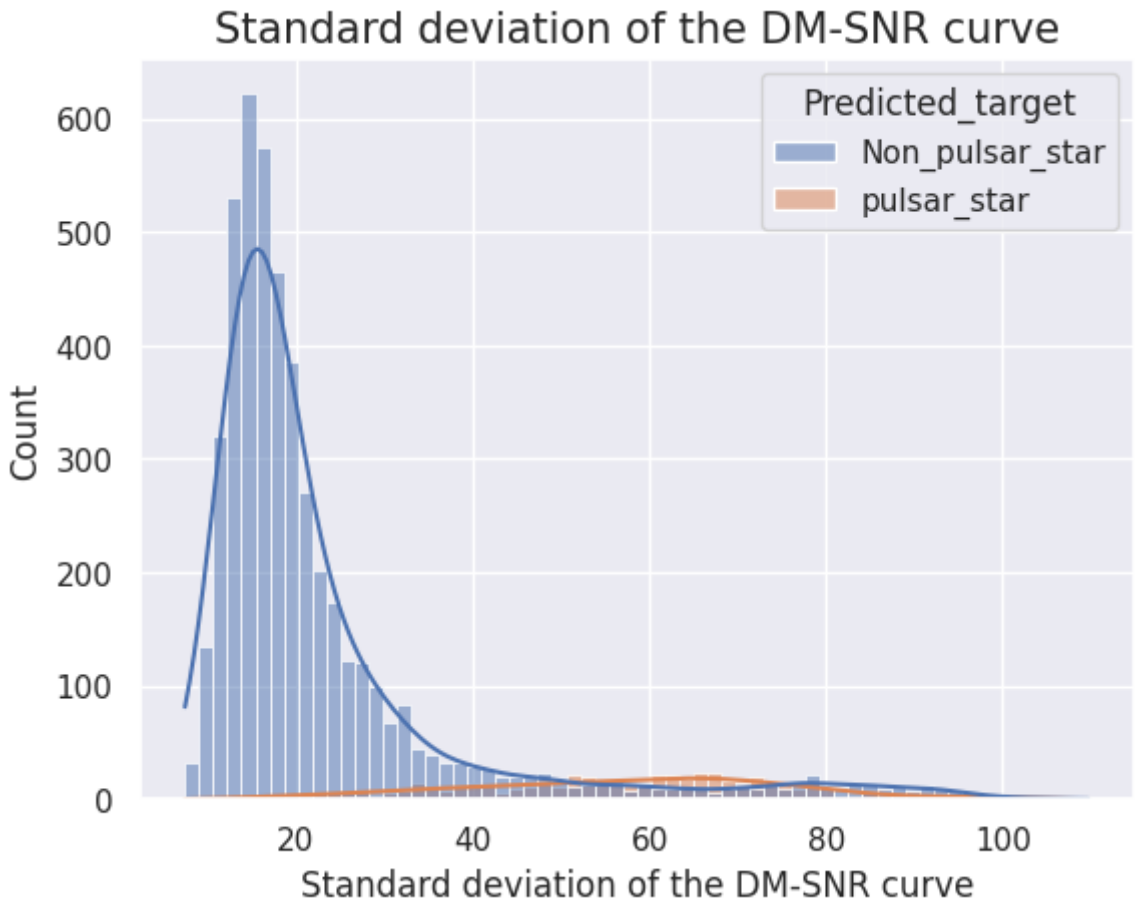
In [ ]:
```
Plot_dist(New_data_test)
```

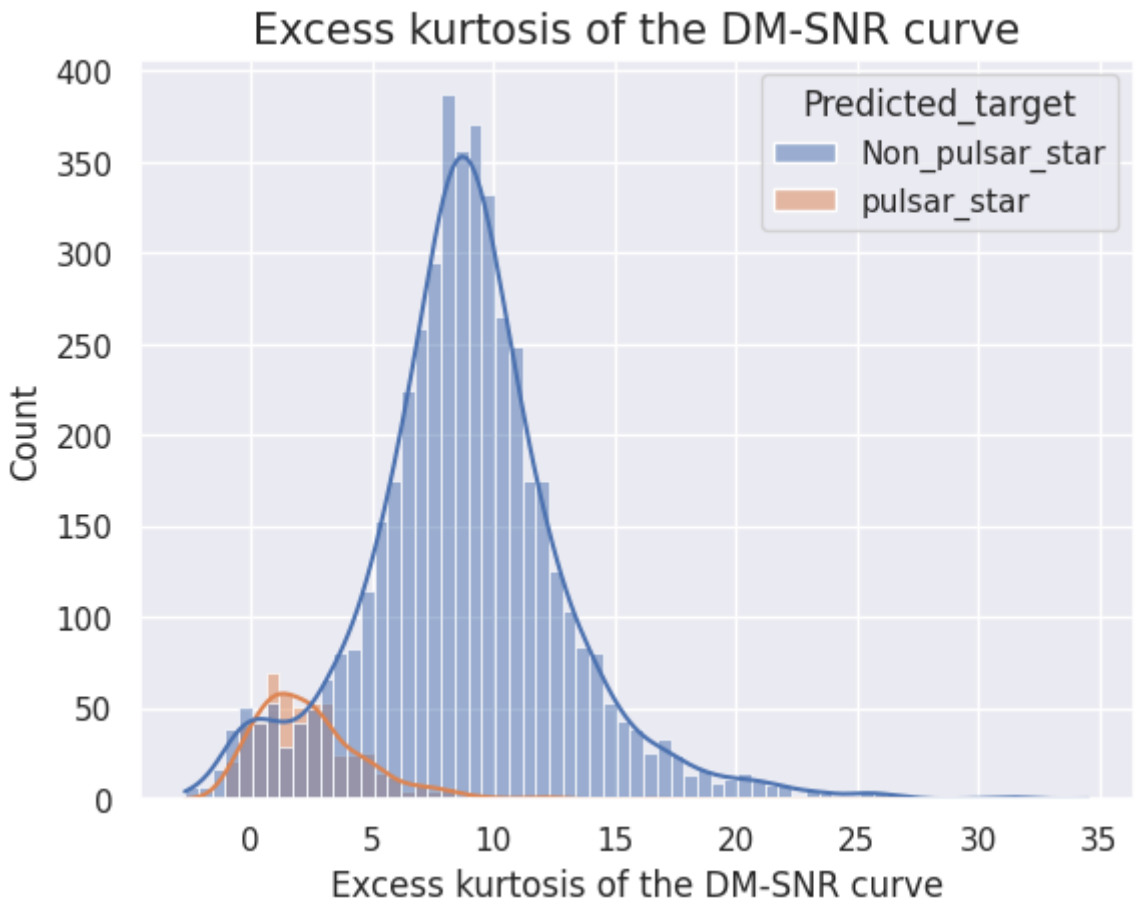

Mean of the integrated profile
Predicted_target
Non_pulsar_star
pulsar_star
Count
0
50
100
150
200
250
300
350
0
25
50
75
100
125
150
175
200
Mean of the integrated profile

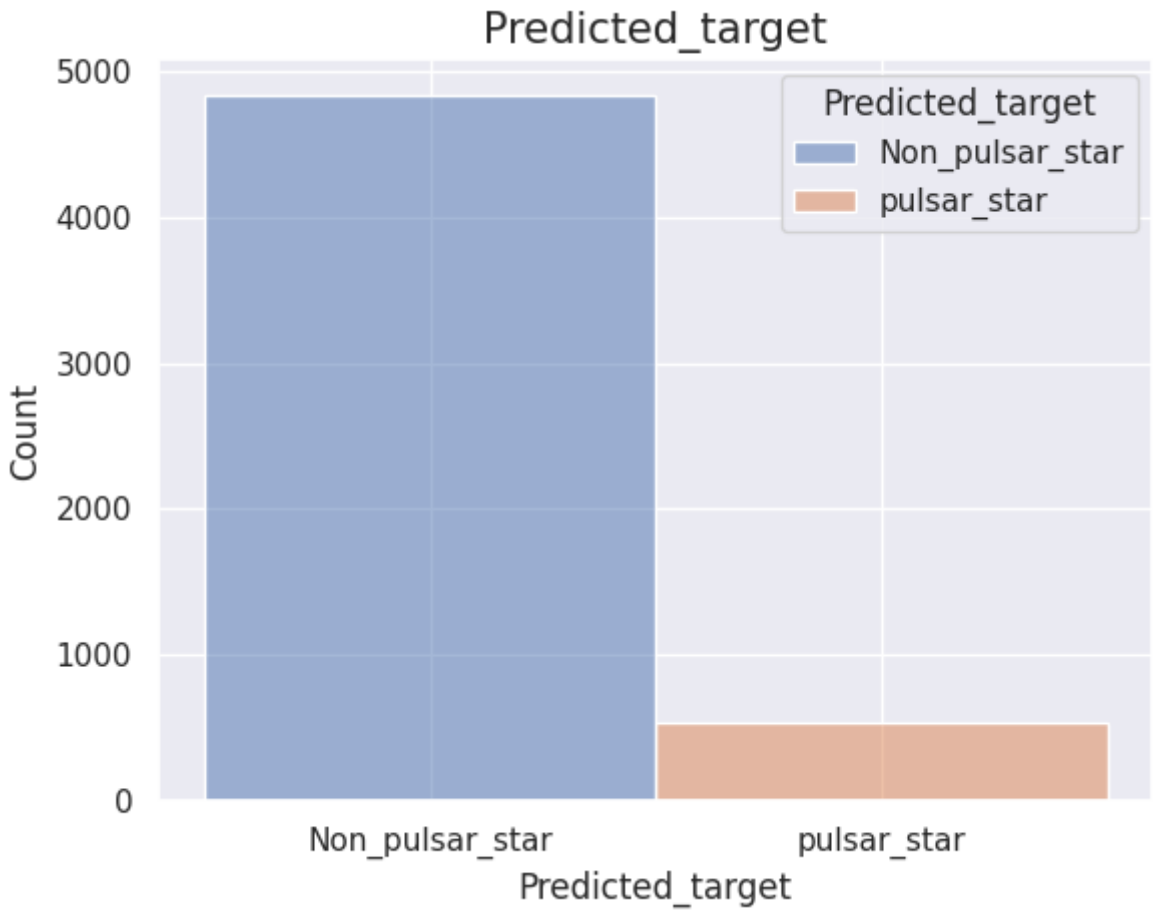

Standard deviation of the integrated profile
Predicted_target
Non_pulsar_star
pulsar_star
Count
0
50
100
150
200
250
300
350
30
40
50
60
70
80
90
100
Standard deviation of the integrated profile


Excess kurtosis of the integrated profile
Predicted_target
Non_pulsar_star
pulsar_star
Count
350
300
250
200
150
100
50
0
−2
0
2
4
6
8
Excess kurtosis of the integrated profile


Mean of the DM-SNR curve
Predicted_target
Non_pulsar_star
pulsar_star
Count
700
600
500
400
300
200
100
0
0
50
100
150
200
Mean of the DM-SNR curve


Standard deviation of the DM-SNR curve
Predicted_target
Non_pulsar_star
pulsar_star
Count
0
100
200
300
400
500
600
20
40
60
80
100
Standard deviation of the DM-SNR curve


Excess kurtosis of the DM-SNR curve
Predicted_target
Non_pulsar_star
pulsar_star
Count
0
50
100
150
200
250
300
350
400
0
5
10
15
20
25
30
35
Excess kurtosis of the DM-SNR curve



End of the code