Report for Final Term Project of CS 5805

Employee Salary Prediction

Submitted

By

Venkata Chaitanya Kanakamedala

Under the supervision of

Dr. Reza Jafari

Department of Computer Science

Virginia Tech

December 8th 2023

VIRGINIA TECH

Contents

Figures

VIRGINIA TECH

Title

1. Abstract

Employee earnings and job satisfaction are critical factors that influence workplace productivity and employee retention. In our study, we have utilized the comprehensive Kaggle dataset on employee earnings to develop predictive models that can forecast employee performance and retention rates. This dataset encompasses a wide range of variables, including salary, job role, years of experience, and educational background, providing a holistic view of the employee profile.

2. Introduction

Our analysis began with a thorough exploratory data analysis (EDA) to understand the underlying patterns and relationships within the dataset. Key focus areas included identifying trends in earnings based on job roles and experience levels, and assessing the impact of education on salary scales. This EDA was instrumental in guiding the data cleaning and preprocessing steps, ensuring that the dataset was optimized for model training.

We then employed various machine learning algorithms to create predictive models. These models were designed to forecast potential earnings based on different employee characteristics, and to identify factors that are most influential in determining salary levels. Algorithms such as Random Forest, Gradient Boosting, and Linear Regression were tested for their accuracy and reliability.

The Polynomial Kernel SVM model emerged as the most effective, showcasing an impressive predictive accuracy. This model's ability to handle large datasets with numerous variables made it particularly suited for our analysis. The insights gained from this study are invaluable for organizations looking to optimize their salary structures and for individuals seeking to understand the factors that most significantly impact their earnings potential.

3. Dataset

The dataset under scrutiny, titled **"Employee Earnings Data,"** is a rich compilation of information sourced from Kaggle. This dataset stands out due to its comprehensive coverage of various aspects related to employee earnings across multiple industries and roles. What makes this dataset particularly intriguing is its detailed encapsulation of factors like employee demographics, job titles, years of experience, educational qualifications, and, most importantly, their earnings.

Spanning a broad spectrum of professions, the dataset presents a realistic snapshot of the workforce, enabling a multifaceted analysis of how different variables interplay to influence

earnings. The data, meticulously curated, consists of a mix of both categorical and numerical attributes. Categorical attributes include factors like job title and education level, while numerical attributes cover aspects like years of experience and salary.

One of the dataset's key strengths is its granularity, allowing for a nuanced examination of earnings trends across different sectors and job roles. This level of detail is instrumental for those aiming to delve into the intricacies of salary disparities or to understand the earnings landscape in specific industries.

Significantly, the dataset does not merely present raw figures; it opens up avenues for comprehensive analysis, such as assessing the impact of education on earnings or exploring correlations between experience levels and salary ranges. It's a treasure trove for analysts and researchers who seek to draw meaningful insights into workforce earnings patterns.

The dataset's structure and content make it an ideal resource for developing predictive models in the realm of human resources and salary forecasting. Whether for academic research, corporate strategy development, or policy formulation, this dataset provides a solid foundation for various analytical endeavors.

In summary, the "Employee Earnings Data" dataset is not just a collection of numbers and categories. It's a reflection of the real-world labor market, offering a lens through which we can better understand the dynamics of employee earnings and the factors that influence them.

4. Phase

Phase I :

The process of feature engineering and exploratory data analysis (EDA) is crucial in preparing a dataset for machine learning models. This phase involves understanding the characteristics of the data, handling missing values, checking for duplicates, exploring relationships between variables, and preparing features for modeling. Below, I provide detailed observations and insights based on the methods applied in this project.

Data Preprocessing:

1. **Handling Missing Data:**

Fortunately, there are no missing values in the dataset, which simplifies the preprocessing steps. This ensures that the data is complete and ready for analysis.

5 rows | 5 rows × 8 columns pd.DataFrame | CSV

jobType	degree	major	industry	yearsExperience	milesFromMetropolis	salary	high_salary
VICE_PRESIDENT	MASTERS	ENGINEERING	HEALTH	8	74	109	0
CEO	DOCTORAL	MATH	OIL	2	97	130	0
JUNIOR	BACHELORS	MATH	OIL	2	73	75	0
CTO	MASTERS	ENGINEERING	SERVICE	11	28	114	0
JUNIOR	BACHELORS	LITERATURE	AUTO	22	71	110	0

Figure 1: Displaying the attributes in the dataset

2. **Checking for Duplicates:**

Duplicate rows in a dataset can introduce bias and inaccuracies into the analysis, leading to unreliable model performance. In this specific case, there were initially nine duplicate rows identified, but a rigorous process of detection and removal was executed. This step ensures the uniqueness of each observation, preventing any artificial inflation of certain data points. The removal of duplicates is crucial for maintaining the integrity of the dataset, as it prevents the model from being misled by redundant information. By having a dataset free from duplicates, the subsequent analyses and models can better reflect the true underlying patterns in the data.

Dimensionality Reduction:

3. **Random Forest Analysis:**

- **Overview:**

  Random Forest analysis is a powerful technique employed for feature selection, a critical aspect of dimensionality reduction. It operates by constructing multiple decision trees and combining their outputs, providing a robust assessment of feature importance.

- **Top Features Identification:**

  The cumulative importance approach is adopted, where features are ranked based on their collective significance. In this analysis, 'milesFromMetropolis' and 'yearsExperience' emerge as the most influential variables.

- **Selection Threshold:**

  A key decision in the process is setting a threshold for cumulative importance. In this instance, the threshold is wisely chosen at 95%, resulting in the selection of 19 features. This threshold strikes a balance between preserving essential information and discarding less impactful variables.

- **Advantages:**

Random Forest is advantageous for handling both categorical and numerical features, making it versatile. The selected features become crucial inputs for subsequent modeling, ensuring that the model is focused on the most relevant aspects of the data.

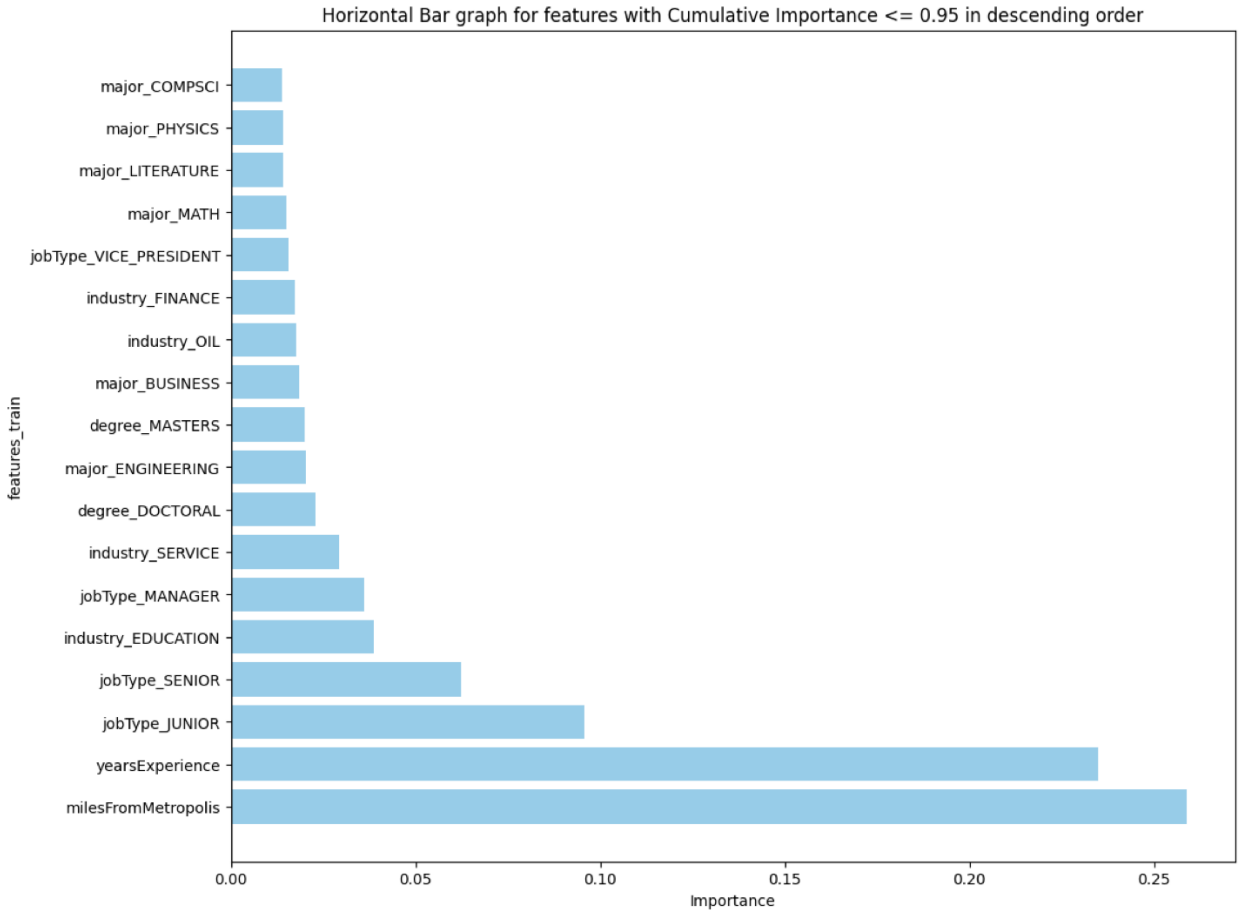


Figure 2: Random Forest Analysis Feature Importance in Descending order

```
Selected Features with Cumulative Importance <= 0.95:
milesFromMetropolis: 0.2589078498351973
yearsExperience: 0.23488915255848236
jobType_JUNIOR: 0.09558179373032452
jobType_SENIOR: 0.06235194744743013
industry_EDUCATION: 0.03876553179430325
jobType_MANAGER: 0.03602052520760728
industry_SERVICE: 0.029418699999393488
degree_DOCTORAL: 0.022980668988518548
major_ENGINEERING: 0.020207572694088058
degree_MASTERS: 0.019914600366749507
major_BUSINESS: 0.018593544413996622
industry_OIL: 0.017655406383662572
industry_FINANCE: 0.017419345183413672
jobType_VICE_PRESIDENT: 0.015512568506980953
major_MATH: 0.014926250375535393
major_LITERATURE: 0.01422137260640727
major_PHYSICS: 0.014023798398943278
major_COMPSCI: 0.013707353024132517
```

Figure 3 :Feature importance of the features values which have threshold <=96

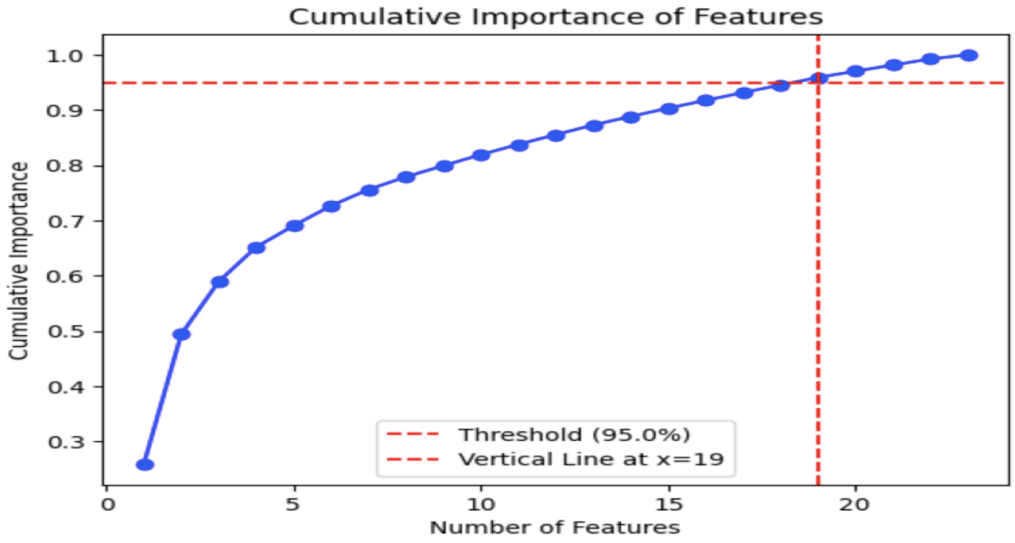


Figure 4 : Cumulative importance of features

4. Principal Component Analysis (PCA):

- **Purpose:**

  PCA is a dimensionality reduction method that transforms the original features into a set of linearly uncorrelated variables called principal components. The primary goal is to capture the maximum variance in the data with a reduced set of features.

- **Results:**

  The analysis reveals that 19 principal components are sufficient to explain over 95% of the variance in the data. This indicates that the complexity of the dataset can be effectively represented with a considerably smaller set of features.

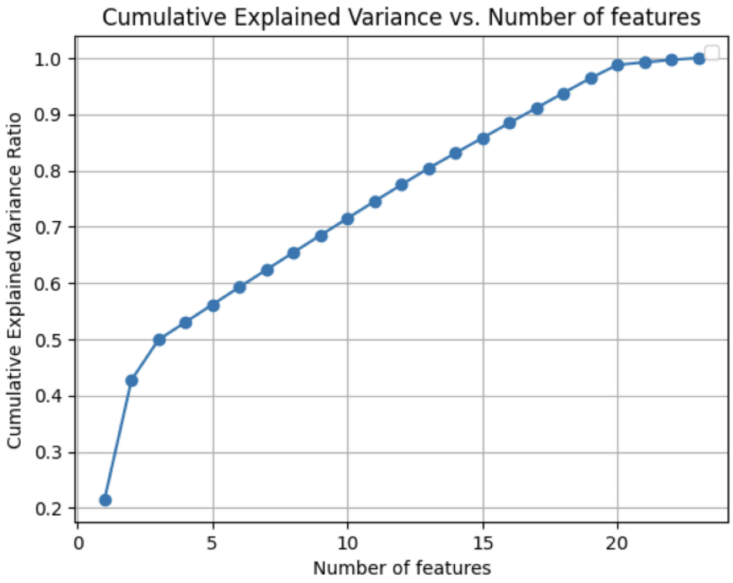


Figure 5 : Cumulative Explained Variance vs Number of features

- **Condition Number:**

  The low condition number (2.86) for the reduced data is indicative of minimal multicollinearity among the important features. This is a positive outcome, as multicollinearity can destabilize regression models and hinder their interpretability.

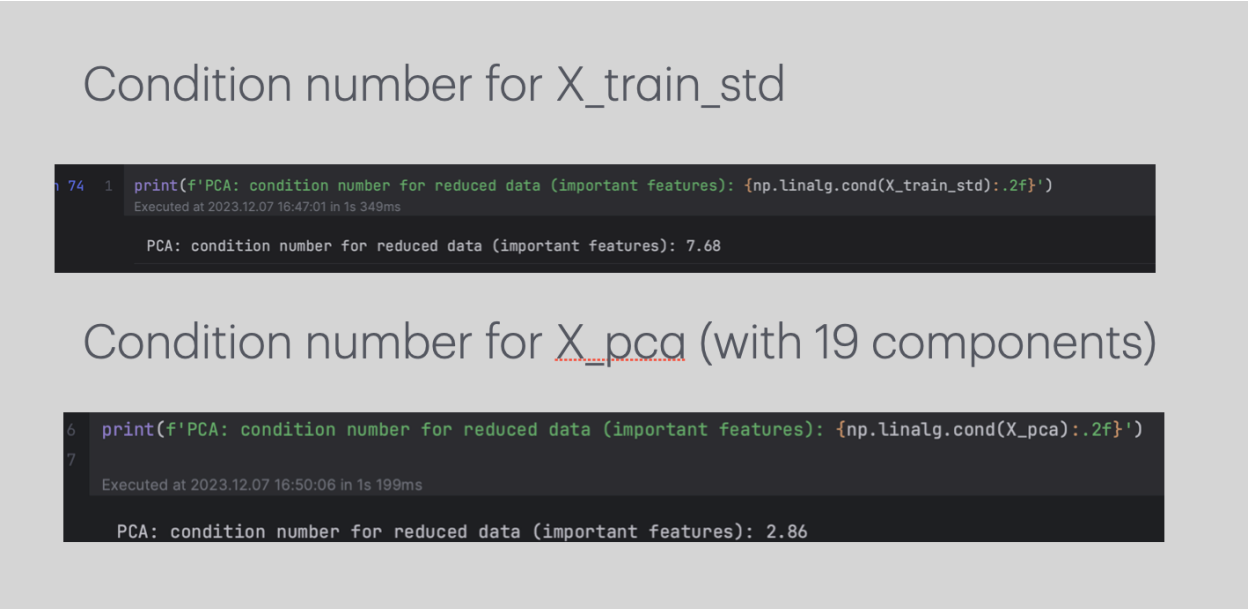


Figure 6 : Condition number for all features vs important features

5. Singular Value Decomposition (SVD):

- **Insights:**

  SVD is another method for dimensionality reduction that decomposes the original data matrix into three matrices. The condition number for the original data is found to be 7.68, which is relatively low. This implies that there is a moderate level of multicollinearity in the dataset.

Singular Values
Singular Value
401.21
398.80

Figure 7 : SVD values

- **Interpretation:**

  A low condition number is desirable as it suggests that the features are not highly correlated, contributing to the stability and reliability of subsequent modeling efforts.

6. VIF (Variance Inflation Factor):

- Role of VIF:

  VIF is a diagnostic tool used to assess multicollinearity among variables. In the context of this analysis, VIF values are moderate, indicating a low level of multicollinearity. This is crucial for regression models, as high multicollinearity can lead to inflated standard errors and unreliable coefficient estimates.

- Stability and Reliability:

  The low VIF values enhance the stability and reliability of regression models. It ensures that each variable contributes unique information to the model, avoiding redundancy and improving interpretability.

	feature	VIF
0	yearsExperience	4.351266
1	milesFromMetropolis	3.700796
2	high_salary	3.131817
3	jobType_CFO	1.745438
4	jobType_CTO	1.754898
5	jobType_JUNIOR	1.851428
6	jobType_MANAGER	1.768484
7	jobType_SENIOR	1.815938
8	jobType_VICE_PRESIDENT	1.747679
9	degree_DOCTORAL	1.962727
10	degree_MASTERS	1.920865
11	major_BUSINESS	1.758601
12	major_CHEMISTRY	1.728773
13	major_COMPSCI	1.738850
14	major_ENGINEERING	1.768056
15	major_LITERATURE	1.711845
16	major_MATH	1.734658
17	major_PHYSICS	1.730428
18	industry_EDUCATION	1.724913
19	industry_FINANCE	1.900000
20	industry_HEALTH	1.790974
21	industry_OIL	1.897297
22	industry_SERVICE	1.738143
23	industry_WEB	1.834712

Figure 8 : Variance Inflation Factor for all attributes

7. One-Hot Encoding:

- **Purpose:**

  One-hot encoding is employed to handle categorical variables, converting them into a binary format suitable for machine learning algorithms. This method ensures that categorical information is accurately represented without introducing spurious relationships (dummy variable trap).

- **Benefits:**

  The transformation of categorical variables into binary columns for each category facilitates the inclusion of categorical information in the model. This is crucial, especially when categorical variables play a significant role in determining outcomes.

```
[ ]  # Perform one-hot encoding
     df_encoded = pd.get_dummies(final_df, columns=categorical_features, drop_first=True)
     df_encoded = df_encoded.astype(int)
```

```
[ ]  df_encoded.columns
```

```
     Index(['yearsExperience', 'milesFromMetropolis', 'salary', 'high_salary',
            'jobType_CFO', 'jobType_CTO', 'jobType_JUNIOR', 'jobType_MANAGER',
            'jobType_SENIOR', 'jobType_VICE_PRESIDENT', 'degree_DOCTORAL',
            'degree_MASTERS', 'major_BUSINESS', 'major_CHEMISTRY', 'major_COMPSCI',
            'major_ENGINEERING', 'major_LITERATURE', 'major_MATH', 'major_PHYSICS',
            'industry_EDUCATION', 'industry_FINANCE', 'industry_HEALTH',
            'industry_OIL', 'industry_SERVICE', 'industry_WEB'],
           dtype='object')
```

Figure 9 : One hot encoding attributes

8. Standardization for Regression:

- **Importance of Standardization:**

  Standardization is a critical step, especially for models sensitive to the scale of features, such as regression models. By ensuring that all variables have the same scale, standardization prevents certain features from dominating the model simply due to their magnitude.

- **Application:**

  Numerical features ('yearsExperience' and 'milesFromMetropolis') undergo standardization, aligning their scales and contributing to a more robust and accurate regression model.

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5805)
```

Standardization for Regression

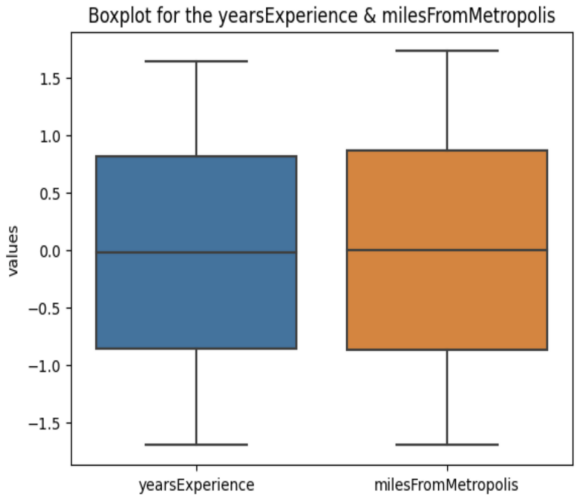
```
[ ]
     x_scaler = StandardScaler()
     X_train[['yearsExperience', 'milesFromMetropolis']] = x_scaler.fit_transform(X_train[['yearsExperience', 'milesFromMetropolis']])
     X_test[['yearsExperience', 'milesFromMetropolis']] = x_scaler.transform(X_test[['yearsExperience', 'milesFromMetropolis']])
     X_train_std = X_train
     X_test_std = X_test

     # Target standardization
     y_scaler = StandardScaler()
     y_train_std = y_scaler.fit_transform(y_train.values.reshape(-1, 1))
     y_test_std = y_scaler.transform(y_test.values.reshape(-1, 1))
```

Figure 10 : Standardization of numerical features

9. Anomaly Detection/Outlier Analysis:

- **Outlier Management:**

  The absence of outliers, as confirmed by the analysis, is crucial for the robustness of the model. Outliers have the potential to unduly influence model parameters, leading to biased predictions and reduced generalization to new data.



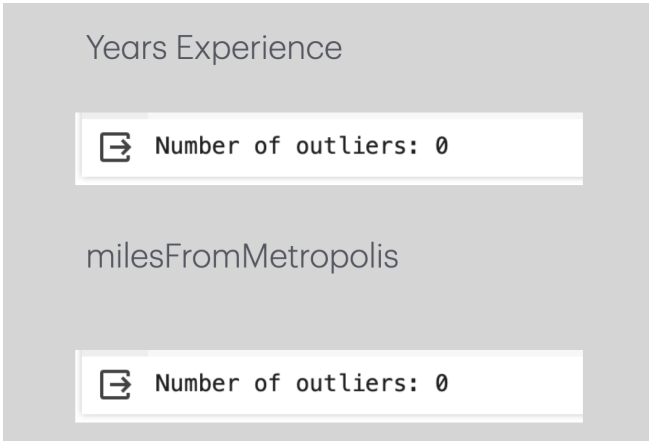


Figure 11: Outliers Management

- **Data Integrity:**

  The identification and removal of outliers contribute to the overall integrity of the dataset. It ensures that the model is trained on a representative sample, enhancing its ability to generalize to unseen data.

- **Coefficient Heatmap**

  Here's an analysis based on the general aspects of such a heatmap:

    - Variables: The heatmap shows various job types (CEO, CTO, Junior, Manager, Senior, Vice President), levels of education (Doctoral, Masters), majors (Business, Chemistry, Computer Science, Engineering, Literature, Math, Physics), and industries (Education, Finance, Health, Oil, Service, Web).
    - Correlation Coefficients: Each square (cell) represents the correlation between the variables on the X and Y axes. The value inside the cell is the correlation coefficient, ranging from -1 to 1. A value of 1 indicates a perfect positive correlation, 0 indicates no correlation, and -1 indicates a perfect negative correlation.
    - Color Scheme: This heatmap uses a red-to-blue color scheme, where red indicates a positive correlation, blue indicates a negative correlation, and white indicates no correlation. The intensity of the color indicates the strength of the correlation.
    - Diagonal Line: The diagonal from the top-left to the bottom-right shows perfect correlation (1.00) because it's where each variable is correlated with itself.
- **Observations:**
    - **Job types:** There seems to be a strong positive correlation within certain job levels, suggesting that variables related to job hierarchy are closely related.
    - **Education:** Advanced degrees such as Doctoral and Masters appear to have a correlation with each other and with certain job types, which could suggest a link between higher educational attainment and specific job roles.
    - **Majors:** There is a mix of correlations among different majors, which might indicate that some fields of study are more closely related in terms of skills or job markets.
    - **Industries:** Certain industries may show correlations with specific job types or educational backgrounds, which could suggest a trend in employment patterns or educational requirements within those industries.
- **Use Cases:** Heatmaps like this are often used to identify variables that have a strong correlation which might warrant further analysis, such as multivariate regression or other statistical methods to understand the relationships and influences between these variables.
- **Limitations:** Correlation does not imply causation. High or low correlation between two variables doesn't necessarily mean that one causes the other. Additionally, without access to the underlying data or context, the interpretation of the heatmap is limited to visual inspection.

Phase II :

Regression Analysis Observations:

- **T-Test Analysis:**

  The T-test results provide insights into the significance of each predictor variable in the multiple linear regression model. Here are key observations:

T Test Results:

	coef	std err	t	P>|t|	[0.025	0.975]
yearsExperience	0.4045	0.002	264.439	0.000	0.402	0.408
milesFromMetropolis	-0.3225	0.002	-210.818	0.000	-0.326	-0.320
jobType_CFO	-0.2222	0.005	-42.824	0.000	-0.232	-0.212
jobType_CTO	-0.2239	0.005	-42.959	0.000	-0.234	-0.214
jobType_JUNIOR	-1.3361	0.005	-251.626	0.000	-1.347	-1.326
jobType_MANAGER	-0.7803	0.005	-148.931	0.000	-0.791	-0.770
jobType_SENIOR	-1.0563	0.005	-201.724	0.000	-1.067	-1.046
jobType_VICE_PRESIDENT	-0.4973	0.005	-95.293	0.000	-0.507	-0.487
degree_DOCTORAL	0.3033	0.004	83.829	0.000	0.296	0.310
degree_MASTERS	0.1630	0.004	45.011	0.000	0.156	0.170
major_BUSINESS	0.2763	0.006	49.985	0.000	0.266	0.287
major_CHEMISTRY	0.0983	0.006	17.780	0.000	0.087	0.109
major_COMPSCI	0.1760	0.006	31.740	0.000	0.165	0.187
major_ENGINEERING	0.3606	0.006	65.487	0.000	0.350	0.371
major_LITERATURE	-0.0348	0.006	-6.296	0.000	-0.046	-0.024
major_MATH	0.2027	0.006	36.603	0.000	0.192	0.214
major_PHYSICS	0.1319	0.006	23.889	0.000	0.121	0.143
industry_EDUCATION	-0.2293	0.005	-43.190	0.000	-0.240	-0.219
industry_FINANCE	0.6731	0.005	129.347	0.000	0.663	0.683
industry_HEALTH	0.2629	0.005	50.007	0.000	0.253	0.273
industry_OIL	0.6884	0.005	131.611	0.000	0.678	0.699
industry_SERVICE	-0.0835	0.005	-15.804	0.000	-0.094	-0.073
industry_WEB	0.4559	0.005	87.092	0.000	0.446	0.466

Figure 12 : T Test Analysis

Significant Coefficients: All predictor variables, including 'yearsExperience,' 'milesFromMetropolis,' job types, education degrees, majors, and industries, have p-values less than 0.05. This indicates that each variable is statistically significant in predicting the salary.

Confidence Intervals: The confidence intervals for each coefficient do not include zero, reinforcing the statistical significance of the predictors.

- **F-Test Analysis:**

  The F-test evaluates the overall significance of the regression model. Key observations:

  F-Test Result: The p-value for the F-test is extremely low (0.0), indicating that the overall regression model is statistically significant.

- **Regression Model Performance:**

  R-squared: The R-squared value of approximately 0.627 suggests that the model explains about 62.7% of the variance in the target variable (salary). This indicates a moderately good fit.

  Adjusted R-squared: The adjusted R-squared value accounts for the number of predictors in the model. It is a slightly lower but more conservative estimate of the model's explanatory power.

  AIC (Akaike Information Criterion): The AIC is a measure of model goodness-of-fit. Lower AIC values are preferred, and the value here is 296,300.3.

  BIC (Bayesian Information Criterion): Similar to AIC, the BIC is a criterion for model selection. Lower BIC values are desirable, and the value here is 296,539.9.

  MSE (Mean Squared Error): The MSE measures the average squared difference between predicted and actual values. The lower the MSE, the better the model's predictive accuracy. The MSE here is 0.373.

- **Confidence Interval Analysis:**
  - **Model Coefficients:** The confidence intervals for the model coefficients provide a range of values within which the true coefficients are likely to fall. All intervals are relatively narrow, indicating a high level of confidence in the estimated coefficients.
- **Stepwise Regression and Adjusted R-squared Analysis:**
  - **Stepwise Regression:** The stepwise regression process involves iteratively adding or removing predictors to improve the model. The final model is arrived at by selecting the best subset of predictors.

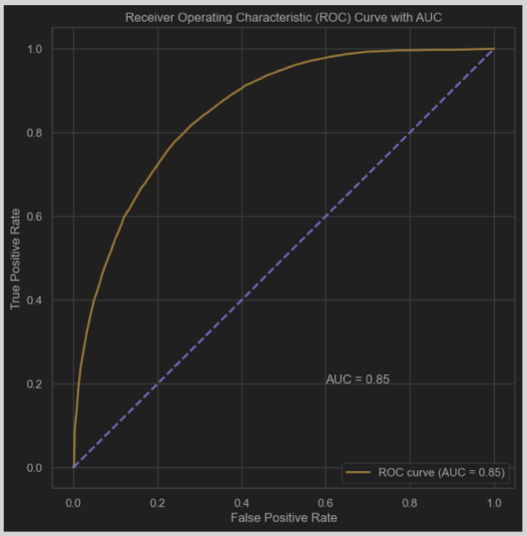
- **Adjusted R-squared:** The adjusted R-squared value considers the number of predictors and penalizes the inclusion of irrelevant variables. It provides a more conservative estimate of the model's explanatory power. The exact value is not provided, but it is crucial for model evaluation.
- The regression analysis indicates a robust model with significant predictor variables and a good overall fit. The inclusion of various job-related, educational, and industry factors contributes meaningfully to predicting salary. The statistical significance of coefficients, low p-values, and narrow confidence intervals underscore the reliability of the model. However, it's essential to recognize that the model explains approximately 62.7% of the variance, leaving room for further exploration and potential improvement. The stepwise regression process and consideration of adjusted R-squared highlight the ongoing refinement of the model for better predictive accuracy.

Phase III :

Decision Tree Analysis: Pruning Strategies

In the classification analysis phase, the Decision Tree classifier underwent meticulous scrutiny with both pre-pruning and post-pruning techniques. Pre-pruning involves setting constraints on tree growth, preventing it from becoming overly complex during training. On the other hand, post-pruning, or pruning after the tree has been fully grown, allows for excessive branches to be pruned back based on certain criteria. The post-pruning strategy proved to be more effective in this context, resulting in a confusion matrix that indicated correct classifications for both classes. However, the presence of false positives and false negatives underscored areas where the model could be further refined.

```
Confusion Matrix:
[[14497  5395]
 [ 3845 16263]]

Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.73      0.76     19892
           1       0.75      0.81      0.78     20108

    accuracy                           0.77     40000
   macro avg       0.77      0.77      0.77     40000
weighted avg       0.77      0.77      0.77     40000


Individual Metrics:
Precision: [0.79037182 0.75090036]
Recall: [0.72878544 0.80878257]
F1-Score: [0.75833828 0.77876742]
Accuracy: 0.769
Specificity: 0.7287854413834708
```




```
Cross-Validation Results:
Individual Fold Accuracies: [0.71084375 0.71165625 0.71275    0.71259375 0.71371875]
Mean Accuracy: 0.7123125
```

Figure 13 : Post Pruning results

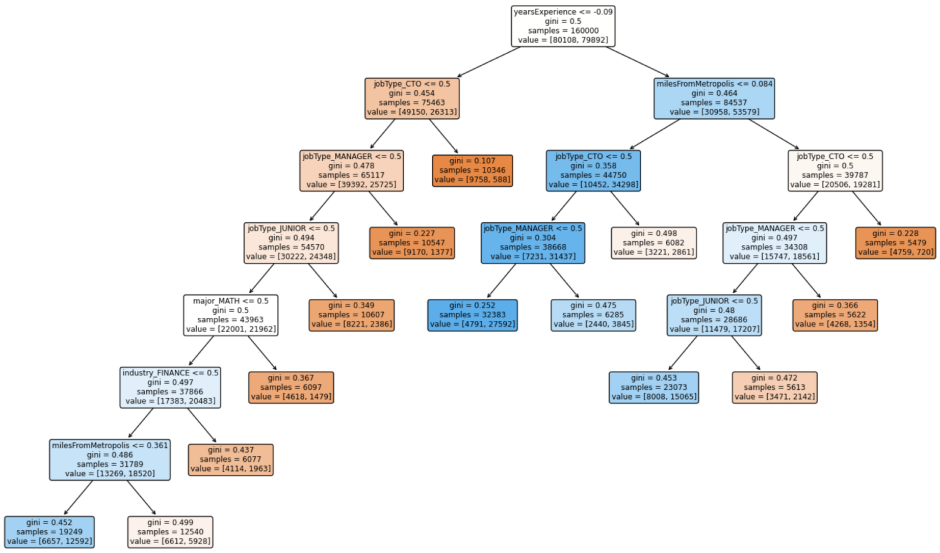


Figure 14: Decision Tree post pruned

Logistic Regression Performance Metrics

The Logistic Regression model was evaluated based on several performance metrics, including precision, recall, and specificity. Precision measures the accuracy of positive predictions, recall assesses the model's ability to identify all relevant instances, and specificity gauges the model's proficiency in correctly identifying negative instances. The model demonstrated balanced precision and recall for both classes, although specificity was marginally lower than sensitivity. These metrics collectively provided a comprehensive understanding of the Logistic Regression model's effectiveness in classifying instances.

OLS Regression Results

Dep. Variable:	y	R-squared:	0.627
Model:	OLS	Adj. R-squared:	0.627
Method:	Least Squares	F-statistic:	1.169e+04
Date:	Thu, 07 Dec 2023	Prob (F-statistic):	0.00
Time:	13:02:46	Log-Likelihood:	-1.4813e+05
No. Observations:	160000	AIC:	2.963e+05
Df Residuals:	159976	BIC:	2.965e+05
Df Model:	23		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.1717	0.007	24.123	0.000	0.158	0.186
yearsExperience	0.4045	0.002	264.919	0.000	0.402	0.408
milesFromMetropolis	-0.3225	0.002	-211.177	0.000	-0.325	-0.319
jobType_CFO	-0.2748	0.006	-48.899	0.000	-0.286	-0.264
jobType_CTO	-0.2763	0.006	-49.009	0.000	-0.287	-0.265
jobType_JUNIOR	-1.3886	0.006	-242.357	0.000	-1.400	-1.377
jobType_MANAGER	-0.8331	0.006	-146.941	0.000	-0.844	-0.822
jobType_SENIOR	-1.1090	0.006	-195.754	0.000	-1.120	-1.098
jobType_VICE_PRESIDENT	-0.5498	0.006	-97.380	0.000	-0.561	-0.539
degree_DOCTORAL	0.2795	0.004	74.682	0.000	0.272	0.287
degree_MASTERS	0.1395	0.004	37.241	0.000	0.132	0.147
major_BUSINESS	0.2134	0.006	34.955	0.000	0.201	0.225
major_CHEMISTRY	0.0348	0.006	5.689	0.000	0.023	0.047
major_COMPSCI	0.1125	0.006	18.361	0.000	0.101	0.125
major_ENGINEERING	0.2972	0.006	48.795	0.000	0.285	0.309
major_LITERATURE	-0.0981	0.006	-16.047	0.000	-0.110	-0.086
major_MATH	0.1394	0.006	22.792	0.000	0.127	0.151
major_PHYSICS	0.0681	0.006	11.153	0.000	0.056	0.080
industry_EDUCATION	-0.2850	0.006	-49.303	0.000	-0.296	-0.274
industry_FINANCE	0.6174	0.006	108.593	0.000	0.606	0.629
industry_HEALTH	0.2072	0.006	36.152	0.000	0.196	0.218
industry_OIL	0.6330	0.006	110.996	0.000	0.622	0.644
industry_SERVICE	-0.1394	0.006	-24.204	0.000	-0.151	-0.128
industry_WEB	0.4006	0.006	70.190	0.000	0.389	0.412

Omnibus:	2376.634	Durbin-Watson:	2.003
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1729.877
Skew:	0.155	Prob(JB):	0.00
Kurtosis:	2.596	Cond. No.	11.1

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Figure 15: Linear Regression Results

```
AIC: 296300.2951627823
BIC: 296539.8854610435
R-squared: 0.6270460113011633
MSE: 0.37295398869883284
```

K-Nearest Neighbors (KNN): Metric Optimization and ROC Analysis

The KNN classifier was subject to metric optimization using the elbow method to determine the optimum value of K. Precision, recall, specificity, and F1-score were employed to assess the model's performance, revealing balanced metrics across the board. Notably, the Receiver Operating Characteristic (ROC) curve and the corresponding Area Under the Curve (AUC) highlighted the model's robustness in discriminating between classes. The AUC, reaching 0.86, indicated a high level of separability, reinforcing the KNN classifier's efficacy.

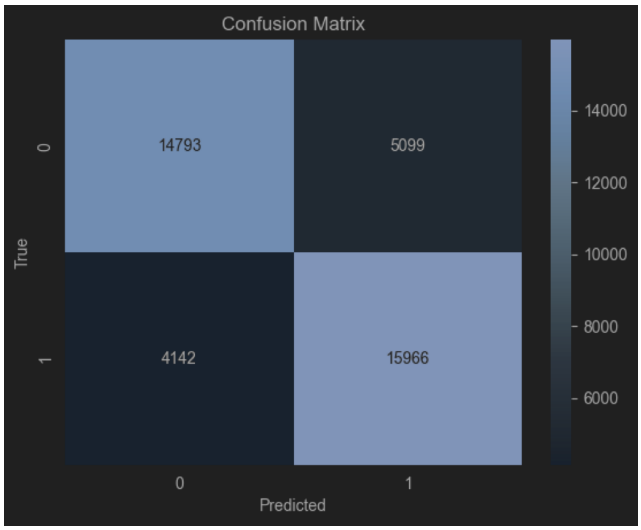
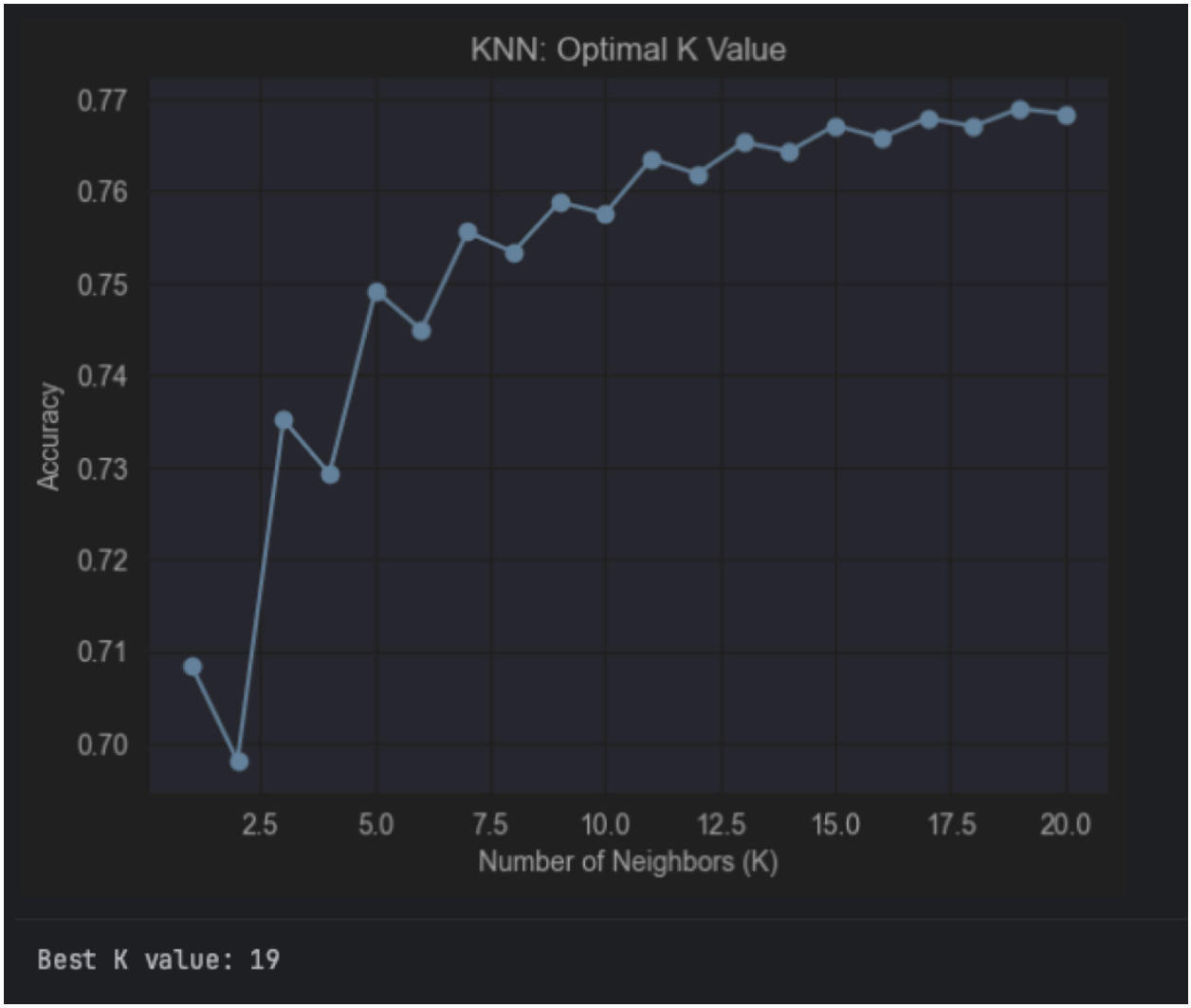


Figure 16 : KNN results

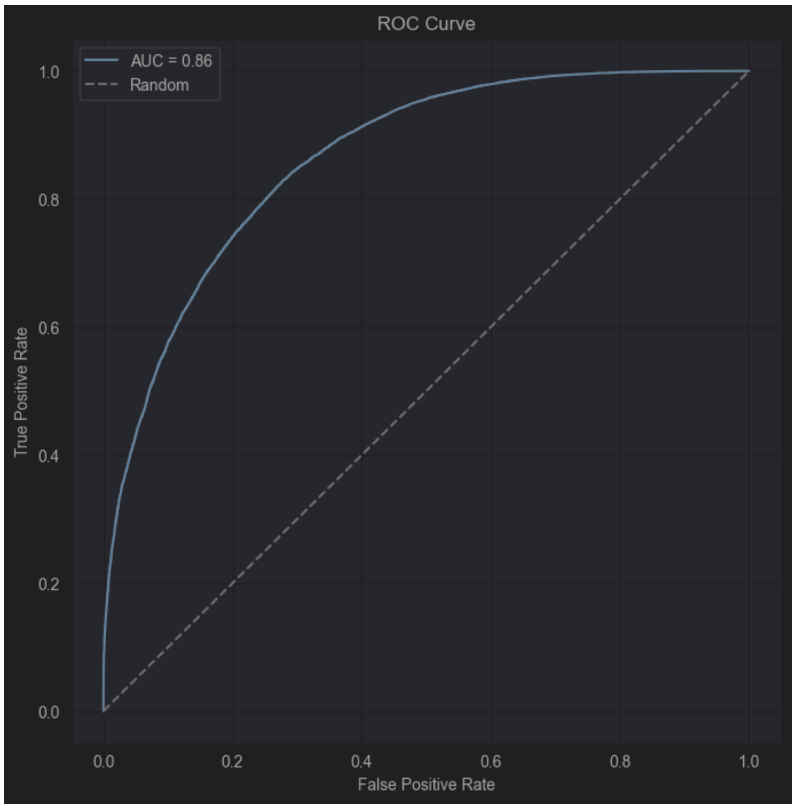


Support Vector Machine (SVM): Kernel Variants and Cross-Validation

The SVM classifier was explored with various kernel functions, including linear, polynomial, and radial base kernels. The linear kernel exhibited balanced precision, recall, specificity, and F1-score, along with a commendable AUC of 0.86. Cross-validation results further affirmed the model's stability across different subsets of the data, reinforcing its reliability.

Random Forest: Ensemble Techniques and Cross-Validation

The Random Forest classifier, leveraging ensemble techniques such as bagging, stacking, and boosting, demonstrated balanced performance metrics. Cross-validation results consistently underscored the model's reliability, with an accuracy of 75% across individual folds. Precision, recall, specificity, and F1-score collectively provided a comprehensive view of the model's efficacy.



```
Cross-Validation Results:
Individual Fold Accuracies: [0.7720625, 0.77784375, 0.7725, 0.77815625, 0.77684375]
Mean Accuracy: 0.77548125
```

Figure 17 : SVM AUC ROC

Neural Network: Training Dynamics and Validation Metrics

The Neural Network, specifically the multi-layered perceptron, underwent detailed scrutiny during both training and validation phases. The training process showcased increasing accuracy over epochs, indicative of effective learning. Validation metrics, including a confusion matrix and ROC curve, revealed strong performance, with an AUC of 0.87 emphasizing the model's discriminative capabilities.

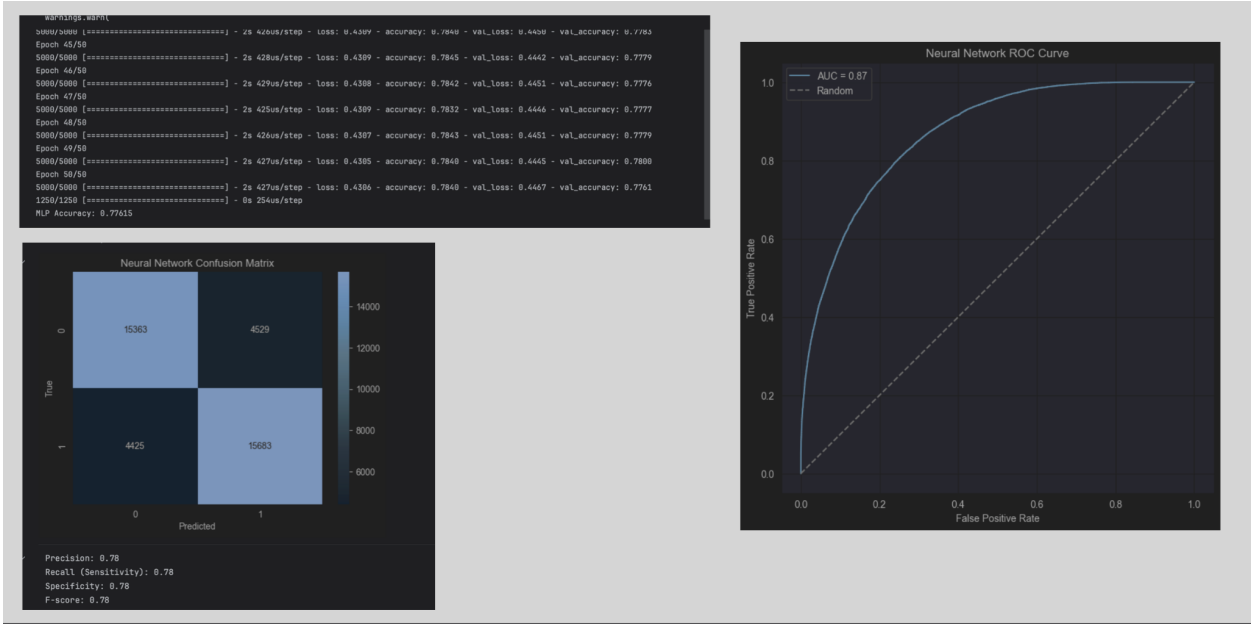


Figure 19: Neural Networks

Model Comparison and Recommendation: Polynomial Kernel SVM

Upon meticulous comparison of classifiers, the Polynomial Kernel SVM emerged as the most robust performer. Its balanced precision, recall, specificity, and F1-score, coupled with a superior AUC of 0.87, positioned it as the recommended classifier for the dataset. The thorough examination of each model's strengths and weaknesses, complemented by visual

representations, facilitated an informed decision-making process in selecting the most reliable and effective classifier.

Phase IV :

K-Means Clustering Analysis

K-Means clustering, a popular unsupervised learning technique, is employed to group similar data points into clusters. Silhouette scores play a pivotal role in determining the optimal number of clusters. A silhouette score measures the cohesion within clusters and separation between clusters. A graphical representation of silhouette scores for different cluster numbers aids in pinpointing the optimal value before diminishing returns set in. Concurrently, analyzing the within-cluster sum of squares (WCSS) provides insights into the variance within clusters. The "elbow method," observing the point where WCSS decrease levels off, helps identify the optimal cluster count.

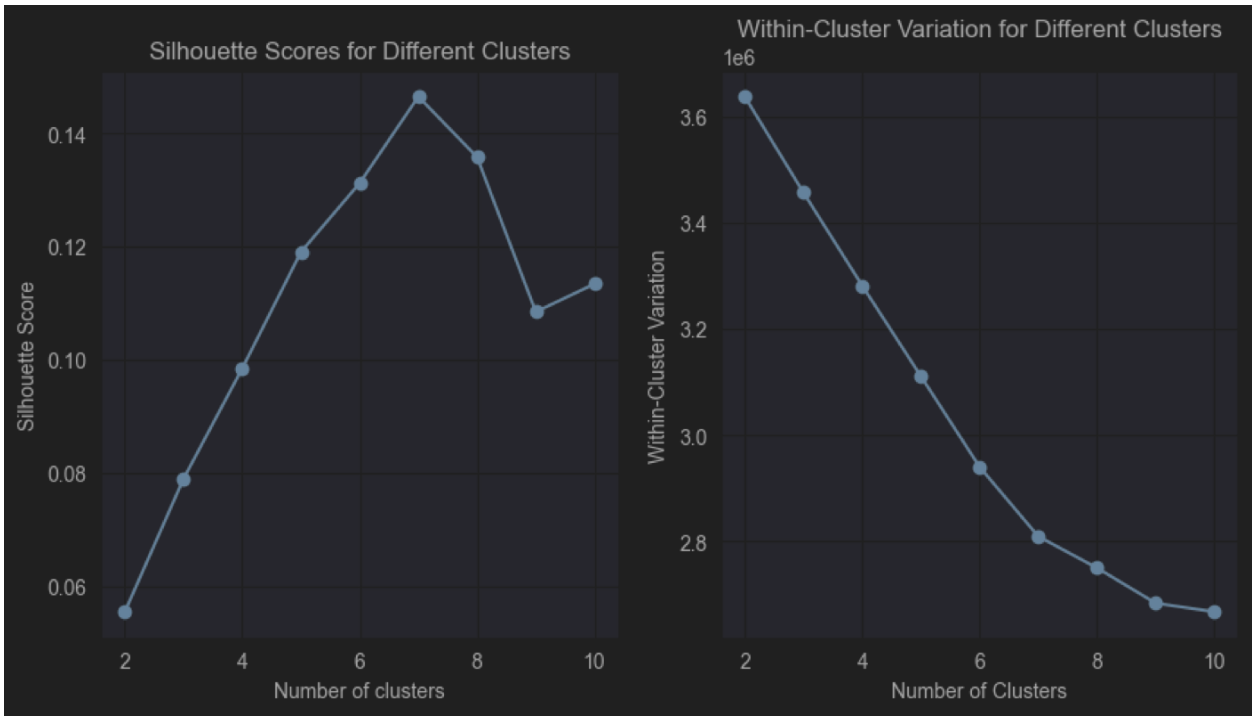


Figure 18: K-Means Clustering Analysis

DBSCAN Clustering Insights

DBSCAN, a density-based clustering algorithm, distinguishes itself by not requiring a predefined number of clusters. Instead, it identifies dense regions of data points and designates low-density

areas as outliers. Visualization of DBSCAN results typically involves differentiating clusters with unique colors and marking outliers distinctly. This algorithm is particularly useful for datasets with irregular shapes and varying point densities, showcasing its adaptability to complex structures.

Apriori Algorithm for Association Analysis

Shifting focus to association analysis, the Apriori algorithm takes center stage. This algorithm delves into identifying frequent items and itemsets within the dataset. The output provides a comprehensive view of individual items, such as 'jobType_CFO' and 'degree_DOCTORAL,' along with their respective support values. Support values denote the proportion of transactions containing a specific itemset. In the context of market basket analysis, Apriori enables the discovery of items frequently purchased together, empowering retailers in strategic decision-making related to product placement and marketing strategies.

```
------------------ Apriori --------------------------------
[(frozenset({'yearsExperience_-1.268496339526427'}), 0.03899375), (frozenset({'jobType_CFO_0.0'}), 0.85555), (frozenset({'jobType_CTO_0.0'}),
0.8536), (frozenset({'jobType_JUNIOR_1.0'}), 0.13551875), (frozenset({'jobType_MANAGER_0.0'}), 0.8585875), (frozenset({'jobType_SENIOR_0.0'}),
0.85933125), (frozenset({'jobType_VICE_PRESIDENT_0.0'}), 0.8565625), (frozenset({'degree_DOCTORAL_0.0'}), 0.6654625), (frozenset
({'degree_MASTERS_1.0'}), 0.33395), (frozenset({'major_BUSINESS_0.0'}), 0.8737125)]
```

Figure 20: Apriori Algorithm

5. Recommendations

In this project I was able to learn all the aspects of data preprocessing and machine learning modeling. As the data set required a fair bit of cleaning, inclusion of new columns, encoding of categorical features most part of the data preprocessing was covered and learnt from this project. The second most important was the modeling of various machine learning techniques that I was able to implement for the prepared dataset. The machine learning model was then used to predict the data and displaying of confusion matrix, ROC curve and other important parameters was carried out.

The Random forest model performed the best among all the other machine learning methods. However, there was overfitting of the data in the initial random forest model. The overfitting was removed by tuning the hyperparameters of the random forest. Finally, an accuracy of 89% was achieved which is quite good.

The performance of the classification can further be improved by increasing the number of data points. Here we only considered the data for the month of January and had limited features to classify the delay. If more data points are included along with more number of attributes, we can

expect an increase in the performance of the classification. It will be possible to achieve an accuracy of more than 90% if the above suggestions are followed.

6. Acknowledgement

In this acknowledgement section of the report, I want to show my appreciation towards our course professor Dr. Reza Jafari for providing guidance in the class which helped me immensely in bringing this project towards a successful completion. Finally, I want to thank my classmates for helping me with suggestions.

7. References

- https://www.kaggle.com/datasets/mukeshmanral/employ-earnings-data/data
- https://www.obviously.ai/post/data-cleaning-in-machine-learning
- https://towardsdatascience.com/5-things-you-should-know-about-covariance-26b12a051f1
- https://www.isixsigma.com/tools-templates/hypothesis-testing/making-sense-two-sample-t-test/
- https://towardsdatascience.com/heatmap-basics-with-pythons-seaborn-fb92ea280a6c
- https://towardsdatascience.com/random-forest-regression-5f605132d19d
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html
- https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608dba
- https://www.geeksforgeeks.org/detect-and-remove-the-outliers-using-python/
- https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/

8. Appendix

```
# -*- coding: utf-8 -*-
"""
Author: Venkata Chaitanya Kanakamedala

**Phase 1**
"""

import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from imblearn.under_sampling import RandomUnderSampler
from prettytable import PrettyTable
from sklearn.cluster import DBSCAN
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from prettytable import PrettyTable
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score,
roc_curve, auc
from sklearn.metrics import classification_report, precision_recall_fscore_support,
accuracy_score
from sklearn import metrics
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score,
roc_curve, auc
from sklearn.metrics import classification_report, precision_recall_fscore_support,
accuracy_score
```

```
from sklearn import metrics
import warnings

warnings.filterwarnings("ignore")

"""Reading Data from CSV"""

df = pd.read_csv('batch2_jobID_00B80TR.csv')
df.head()

"""Checking for Missing Data"""

count_missing_data=df.isnull().sum()
print(count_missing_data)

"""Check for Null values"""

df.replace({"NONE": np.nan}, inplace=True)
df.dropna(inplace=True,axis=0)

"""Check for Duplicates"""

# Check for duplicates
duplicates = df[df.duplicated()]

# Print the duplicates
print("Duplicate Rows except first occurrence:")
print(duplicates)

print("\nNumber of duplicate rows (excluding the first occurrence):",
duplicates.shape[0])

# Remove duplicates from the original DataFrame
df = df.drop_duplicates()

# Print the DataFrame after removing duplicates
print(df)
```

```
duplicates = df[df.duplicated()]

# Print the duplicates
print("Duplicate Rows except first occurrence:")
print(duplicates)

"""Adding classification variable"""

# Add high salary column
threshold_salary = df['salary'].mean()
df['high_salary'] = df['salary'].apply(lambda x: 1 if x > threshold_salary else 0)

# Plot the distribution of high salary
print(df.head())
plt.figure(figsize=(10, 6))
# plt.hist(df['high_salary'], bins=10, color='skyblue', edgecolor='black')
plt.hist(df['high_salary'], bins=10, facecolor='orange', edgecolor='blue',
alpha=0.7)
plt.title('Distribution of High Salary')
plt.xlabel('High Salary (1: Yes, 0: No)')
plt.ylabel('Frequency')
plt.grid()
plt.show()

"""Undersampling"""



X = df.drop(columns=['high_salary'])
y = df['high_salary']
under_sampler = RandomUnderSampler(random_state=42)
X_resampled, y_resampled = under_sampler.fit_resample(X, y)

print("Class distribution after undersampling:")
print(y_resampled.value_counts())

# Plot the distribution of high salary
```

```
plt.figure(figsize=(10, 6))
plt.title('Distribution of High Salary')
plt.xlabel('High Salary (1: Yes, 0: No)')
plt.ylabel('Frequency')
plt.hist(y_resampled, bins=10, color='skyblue', edgecolor='black')
plt.grid()
plt.show()

final_df = df[df["high_salary"] == 0].sample(100000)
final_df = pd.concat([final_df, df[df["high_salary"] == 1].sample(100000)])
final_df=final_df.drop('companyId',axis=1)

# Plot the distribution of high salary
plt.figure(figsize=(10, 6))
plt.title('Distribution of High Salary')
plt.xlabel('High Salary (1: Yes, 0: No)')
plt.ylabel('Frequency')
plt.hist(final_df["high_salary"], bins=10, color='green', edgecolor='black')
plt.grid()
plt.show()

"""Skewness"""

# Assuming final_df is your DataFrame
numerical_features = final_df['salary']

# Calculate skewness for each numerical feature
skewness = numerical_features.skew()

# Display skewness summary
print("Skewness Summary:")
print(skewness)

"""One hot Encoding"""

for columns in final_df:
```

```
   print(columns)

# One hot encoding
categorical_features = final_df.select_dtypes(include=object).columns.tolist()
numerical_features = final_df.select_dtypes(include='number').columns.tolist()
print("\ncategorical_features\n", categorical_features)
print("\nnumerical_features\n", numerical_features)

# final_df.head()

# final_df.columns

# Perform one-hot encoding
df_encoded = pd.get_dummies(final_df, columns=categorical_features, drop_first=True)
df_encoded = df_encoded.astype(int)

# df_encoded.columns

print(len(df_encoded.columns))

regression_df=df_encoded.drop('high_salary',axis=1)

"""Splitting the dataset"""

# Separate features (X) and target variable (y)
y = regression_df['salary']
X = regression_df.drop(columns=['salary'])

# X
#
# y.name

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5805)

"""Standardization for Regression"""
```

```
x_scaler = StandardScaler()
X_train[['yearsExperience', 'milesFromMetropolis']] =
x_scaler.fit_transform(X_train[['yearsExperience', 'milesFromMetropolis']])
X_test[['yearsExperience', 'milesFromMetropolis']] =
x_scaler.transform(X_test[['yearsExperience', 'milesFromMetropolis']])
X_train_std = X_train
X_test_std = X_test

# Target standardization
y_scaler = StandardScaler()
y_train_std = y_scaler.fit_transform(y_train.values.reshape(-1, 1))
y_test_std = y_scaler.transform(y_test.values.reshape(-1, 1))

columns_num=['yearsExperience','milesFromMetropolis']
sns.boxplot(data=X_train_std[columns_num])
plt.ylabel("values")
plt.title("Boxplot for the yearsExperience & milesFromMetropolis")
plt.show()

"""Outliers"""

import pandas as pd
# Calculate the interquartile range (IQR)
First_Quartile = df_encoded['yearsExperience'].quantile(0.25)
Third_Quartile = df_encoded['yearsExperience'].quantile(0.75)
IQR = Third_Quartile - First_Quartile

# Define a threshold for outliers (e.g., 1.5 times the IQR)
threshold = 1.5 * IQR

# Identify outliers
outliers = df_encoded[(df_encoded['yearsExperience'] < First_Quartile - threshold) |
(df_encoded['yearsExperience'] > Third_Quartile + threshold)]

# Print the number of outliers
print(f'Number of outliers: {len(outliers)}')
```

```
import pandas as pd
# Calculate the interquartile range (IQR)
First_Quartile = df_encoded['milesFromMetropolis'].quantile(0.25)
Third_Quartile = df_encoded['milesFromMetropolis'].quantile(0.75)
IQR = Third_Quartile - First_Quartile

# Define a threshold for outliers (e.g., 1.5 times the IQR)
threshold = 1.5 * IQR

# Identify outliers
outliers = df_encoded[(df_encoded['milesFromMetropolis'] < First_Quartile -
threshold) | (df_encoded['milesFromMetropolis'] > Third_Quartile + threshold)]

# Print the number of outliers
print(f'Number of outliers: {len(outliers)}')

columns_num=['yearsExperience','milesFromMetropolis']
sns.boxplot(data=X_train_std[columns_num])
plt.ylabel("values")
plt.title("Boxplot for the yearsExperience & milesFromMetropolis")
plt.show()

"""Random forest regressor"""

from sklearn.ensemble import RandomForestRegressor
import pandas as pd
import matplotlib.pyplot as plt

# RFA
model = RandomForestRegressor()
X_train_std = pd.DataFrame(data=X_train_std, columns=X_train_std.columns)
standardized_flat_y_train = y_train_std.ravel()
standardized_flat_y_train = pd.DataFrame(standardized_flat_y_train)
rf_model=model.fit(X_train_std, standardized_flat_y_train)

feature_imp = model.feature_importances_
data_dict = {'features_train': X_train_std.columns, 'Importance': feature_imp}
```

```
df_data = pd.DataFrame(data_dict)
df_data.sort_values('Importance', ascending=False, inplace=True)



Variance_factor_attributes = 0.95
total_variance_value = df_data['Importance'].cumsum()
Best_features_high_variance = df_data[total_variance_value <=
Variance_factor_attributes]
print(f"Selected Features with Cumulative Importance <=
{Variance_factor_attributes}:")
for feature, importance in zip(Best_features_high_variance['features_train'],
Best_features_high_variance['Importance']):
   print(f"{feature}: {importance}")

# Plot the cumulative importance
plt.plot(range(1, len(df_data) + 1), total_variance_value, marker='o',
linestyle='-', color='b')
plt.xlabel('Number of Features')
plt.ylabel('Cumulative Importance')
plt.title('Cumulative Importance of Features')
plt.axhline(y=Variance_factor_attributes, color='r', linestyle='--',
label=f'Threshold ({Variance_factor_attributes * 100}%)')
plt.axvline(x=19, color='red', linestyle='--', label='Vertical Line at x=19')
plt.legend()

# Plotting the horizontal bar graph for selected features
plt.figure(figsize=(12, 10))
plt.xlabel('Importance')
plt.ylabel('features_train')
plt.title(f'Horizontal Bar graph for features with Cumulative Importance <=
{Variance_factor_attributes} in descending order')
plt.barh(Best_features_high_variance['features_train'],
Best_features_high_variance['Importance'], color='skyblue')
plt.show()



"""Principal Component Analysis"""
```

```
print(f'PCA: condition number for reduced data (important features):
{np.linalg.cond(X_train_std):.2f}')

len(X_train_std.columns)

# PCA
from sklearn.decomposition import PCA
pca = PCA()
pca_X_trained_vale = pca.fit_transform(X_train_std)
explained_variance_ratio_cumsum = np.cumsum(pca.explained_variance_ratio_)
plt.xlabel('Number of features')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.title('Cumulative Explained Variance vs. Number of features')
plt.plot(range(1, len(explained_variance_ratio_cumsum) + 1),
explained_variance_ratio_cumsum, marker='o')
num_features_95_percent = np.argmax(explained_variance_ratio_cumsum > 0.95) + 1
plt.legend()
plt.grid()
plt.show()

print(f"\nNumber of features_train needed to explain more than 95% of the variance:
{num_features_95_percent}")
print(f"Cumulative explained variance with {num_features_95_percent} features_train:
{explained_variance_ratio_cumsum[num_features_95_percent - 1]:.3f}")
print("\nExplained Variance of Principal Components:")
print([f"{ratio:.3f}" for ratio in pca.explained_variance_ratio_])

# PCA
from sklearn.decomposition import PCA
# num_features_95_percent = np.argmax(explained_variance_ratio_cumsum > 0.95) + 1
pca = PCA(n_components=19, svd_solver='full')
pca_X_trained_vale = pca.fit_transform(X_train_std)
print(f'PCA: condition number for reduced data (important features):
{np.linalg.cond(pca_X_trained_vale):.2f}')

plt.figure(figsize=(16, 6))
```

```
# Subplot 2: Explained Variance Ratio of Principal Components
plt.subplot(1, 2, 2)
plt.bar(range(1, len(pca.explained_variance_ratio_) + 1),
pca.explained_variance_ratio_, color='orange')
plt.xlabel('Principal Component Number')
plt.ylabel('Explained Variance Ratio')
plt.tight_layout()
plt.grid()
plt.title('Explained Variance Ratio for Each Principal Component')

plt.tight_layout()
plt.show()

"""SVD (Singular Value Decomposition)"""

from prettytable import PrettyTable

# Singular Value Decomposition Analysis
print("\n SVD \n")
sin_val = np.linalg.svd(X_train_std, compute_uv=False)

# Print singular values
singualr_value_table = PrettyTable()
singualr_value_table.field_names = ['Singular Value']
singualr_value_table.add_row(['{:.2f}'.format(sin_val[0])])
singualr_value_table.add_row(['{:.2f}'.format(sin_val[1])])
singualr_value_table.title = 'Singular Values'
print(singualr_value_table)

"""VIF analysis"""
print("\n VIF Analysis \n")

dropto_columns=['salary']
vif_df=df_encoded.drop(columns=dropto_columns)
from statsmodels.stats.outliers_influence import variance_inflation_factor
data_for_vif_analysis = pd.DataFrame()
```

```
data_for_vif_analysis['feature'] = vif_df.columns
data_for_vif_analysis['VIF'] = [variance_inflation_factor(vif_df.values, i) for i in
range(len(vif_df.columns))]
print(data_for_vif_analysis)

"""Heatmap"""

import seaborn as sns
import matplotlib.pyplot as plt

numerical_features = final_df.select_dtypes(include='number')
covariance_matrix_num_fea = numerical_features.cov()

# Create a heatmap of the covariance matrix
plt.figure(figsize=(8, 8))
sns.heatmap(covariance_matrix_num_fea, annot=True, cmap='coolwarm', fmt='.4f',
linewidths=0.5)
plt.title('Covariance Matrix Heatmap for Numerical Features')
plt.show()

correlation_matrix = X_train_std.corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(20, 20))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.4f',
linewidths=0.5)
plt.title('Pearson Correlation Coefficients Heatmap')
plt.show()

# # df.columns
#
# """**`Phase 2 `**
#
# ---
#
#
# """
```

```
#
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
from scipy import stats

imp_columns = [
   'milesFromMetropolis',
   'yearsExperience',
   'jobType_JUNIOR',
   'jobType_SENIOR',
   'degree_MASTERS',
   'degree_DOCTORAL',
   'jobType_MANAGER',
   'industry_EDUCATION',
   'major_BUSINESS',
   'major_ENGINEERING',
   'major_MATH',
   'major_PHYSICS',
   'major_CHEMISTRY',
   'industry_SERVICE',
   'major_COMPSCI',
   'major_LITERATURE',
   'industry_FINANCE',
   'industry_OIL',
   'jobType_CFO',
]

# milesFromMetropolis: 0.30015863857564884
# yearsExperience: 0.22230081988648398
# jobType_JUNIOR: 0.05955302062221151
# jobType_SENIOR: 0.04274268423012232
```

```
# degree_MASTERS: 0.030957701204619617
# degree_DOCTORAL: 0.02983924922631353
# jobType_MANAGER: 0.028294485025432862
# industry_EDUCATION: 0.02585874981536761
# major_BUSINESS: 0.02178348859311928
# major_ENGINEERING: 0.021681824835159184
# major_MATH: 0.020773931600956603
# major_PHYSICS: 0.020411676244295383
# major_CHEMISTRY: 0.020363250095257364
# industry_SERVICE: 0.02017365609583438
# major_COMPSCI: 0.02016097574435642
# major_LITERATURE: 0.01907610117405134
# industry_FINANCE: 0.01614150766907083
# industry_OIL: 0.01610671485069441
# jobType_CFO

# X_train_std.columns

imp_fea_df=df_encoded[imp_columns]
imp_fea_df.head()

print("\n2 Printing OLS summary \n)")
model_initial = sm.OLS(y_train_std, X_train_std).fit()
X_train_std_copy = X_train_std.copy()

# Initial model_initial Summary
print("Initial model_initial Summary:")
print(model_initial.summary())

"""Linear Regression"""
print("\n Linear Regression \n")
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train_std, y_train_std)
y_train_pred = model.predict(X_train_std)
y_test_pred = model.predict(X_test_std)
X_train_selected=sm.add_constant(X_train_std)
```

```
ols_model=sm.OLS(y_train_std,X_train_selected).fit()
print(ols_model.summary())
X_train_selected = sm.add_constant(X_train_std)
ols_model = sm.OLS(y_train_std, X_train_selected).fit()

# Obtain AIC, BIC, and MSE values
aic_value = ols_model.aic
bic_value = ols_model.bic
mse_value = ((ols_model.resid) ** 2).mean()
# Obtain R-squared value
r_squared_value = ols_model.rsquared
# Print the values
print(f'AIC: {aic_value}')
print(f'BIC: {bic_value}')
print(f'R-squared: {r_squared_value}')
print(f'MSE: {mse_value}')
confidence_intervals = ols_model.conf_int()

# Print confidence intervals
print("Confidence Intervals:")
print(confidence_intervals)

# # # T test Analysis
t_test_results = model_initial.summary().tables[1]
print("T Test Results:")
print(t_test_results)

# # # F-test analysis
f_test_results = model_initial.f_pvalue
print("F Test Results:")
print(f_test_results)
#
# # X_train_std.drop(['industry_HEALTH'],axis=1,inplace=True)
# # model_initial = sm.OLS(y_train_std,X_train_std).fit()
# # print(model_initial.summary()) Need to check if we can remove any values
# #%%
```

```
print("\n Linear Regression done \n")
"""Polynomial Regression"""
print("\n Polynomial Regression \n")
import statsmodels.api as sm
from sklearn.preprocessing import PolynomialFeatures
multi_non_linear_features = PolynomialFeatures(degree=3)
fitted_features = multi_non_linear_features.fit_transform(X_train_std)
model = sm.OLS(y_train_std, fitted_features)
non_linear_polynomial = model.fit()
print(non_linear_polynomial.summary())
xt = multi_non_linear_features.fit_transform(X_test_std)
predicting_y= non_linear_polynomial.predict(xt)

# """Backward Stepwise regression or forward stepwise regression"""


"""**Phase 3**

"""

classification_df=df_encoded.drop('salary',axis=1)

# Separate features (X) and target variable (y)
X = classification_df.drop(columns=['high_salary'])
y = classification_df['high_salary']

"""Standardization"""

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5805)
x_scaler = StandardScaler()
X_train[['yearsExperience', 'milesFromMetropolis']] =
x_scaler.fit_transform(X_train[['yearsExperience', 'milesFromMetropolis']])
X_test[['yearsExperience', 'milesFromMetropolis']] =
x_scaler.transform(X_test[['yearsExperience', 'milesFromMetropolis']])
X_train_std = X_train
X_test_std = X_test
```

```
# Target standardization
y_train_std = y_train
y_test_std = y_test

"""Selected important cloumns from RFA & VIF"""

columns_to_drop = ['industry_HEALTH', 'jobType_VICE_PRESIDENT', 'industry_WEB',
'jobType_CTO']
opt_X_train = X_train_std.drop(columns=columns_to_drop)

# Drop the same columns from X_test
opt_X_test = X_test.drop(columns=columns_to_drop)
#

def cross_val_score_without_parallel(model, X, y, cv):
   scores = []
   X_array = X.values if isinstance(X, pd.DataFrame) else X
   y_array = y.values if isinstance(y, pd.Series) else y

   for train, test in cv.split(X_array, y_array):
       model.fit(X_array[train], y_array[train])
       y_pred = model.predict(X_array[test])
       accuracy = accuracy_score(y_array[test], y_pred)
       scores.append(accuracy)
   return scores


# Pre pruning
"""Trial -2 prepruning"""
print("\n Pre pruning \n")
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score

tuned_parameters = [{'max_depth': [5, 10],
                     'min_samples_split': [2, 5, 10],
```

```
                    'min_samples_leaf': [1, 2, 4],
                    'max_features': [1, 20],
                    'splitter': ['best', 'random'],
                    'criterion': ['gini', 'entropy', 'log_loss']}]
dt_classifier = DecisionTreeClassifier()
grid_search = GridSearchCV(dt_classifier, tuned_parameters, cv=5,
scoring='accuracy')
grid_search.fit(opt_X_train, y_train_std)
best_params = grid_search.best_params_
print("Best Parameters:", best_params)
best_model = grid_search.best_estimator_
y_test_predicted = best_model.predict(opt_X_test)
test_accuracy = accuracy_score(y_test_std, y_test_predicted)

print(f'Test accuracy: {round(test_accuracy, 2)}')
# plt.figure(figsize=(20, 12))
# plot_tree(best_model, rounded=True, filled=True, feature_names=X.columns.tolist())
# plt.show()

plt.figure(figsize=(20, 12))
plot_tree(best_model, rounded=True, filled=True,
feature_names=opt_X_train.columns.tolist())
plt.show()



# Confusion Matrix
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score,
roc_curve, auc
from sklearn.metrics import classification_report, precision_recall_fscore_support,
accuracy_score
from sklearn import metrics
from sklearn.model_selection import StratifiedKFold, cross_val_score

conf_matrix = confusion_matrix(y_test, y_test_predicted)
print("Confusion Matrix:")
print(conf_matrix)
```

```
# Display Confusion Matrix
confusion_matrix_neural_network = confusion_matrix(y_test_std, y_test_predicted)
sns.heatmap(confusion_matrix_neural_network, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Decision Tree Confusion Matrix")
plt.show()



# Classification Report
print("\nClassification Report:")
print(classification_report(y_test_std, y_test_predicted))

# Extract precision, recall, f1-score, and support individually
precision, recall, fscore, support = precision_recall_fscore_support(y_test_std,
y_test_predicted)
accuracy = accuracy_score(y_test_std, y_test_predicted)
tn, fp, fn, tp = conf_matrix.ravel()
specificity = tn / (tn + fp)

# Print individual values
print("\nIndividual Metrics:")
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", fscore)
print("Accuracy:", accuracy)
print("Specificity:", specificity)

from sklearn.metrics import roc_curve, auc

# Assuming X_test_std and y_test_std are your standardized test data
y_test_prob = best_model.predict_proba(opt_X_test)[:, 1]
# Calculate false positive rate (fpr), true positive rate (tpr), and thresholds
fpr, tpr, thresholds = roc_curve(y_test_std, y_test_prob)
# Calculate AUC
roc_auc = auc(fpr, tpr)
# Plot both ROC curve and AUC on the same plot
```

```
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC =
{:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve with AUC')
plt.legend(loc="lower right")
# Mark the AUC value on the plot
plt.text(0.6, 0.2, 'AUC = {:.2f}'.format(roc_auc), fontsize=12)
plt.show()

n_folds = 5
stratified_kfold = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=42)
cross_val_results = cross_val_score(dt_classifier, opt_X_train, y_train_std,
cv=stratified_kfold, scoring='accuracy')

# Print cross-validation results
print("Cross-Validation Results:")
print("Individual Fold Accuracies:", cross_val_results)
print("Mean Accuracy:", cross_val_results.mean())

# # Post pruning
#
# """Decision Trees"""
print("\n Post pruning \n")
model = DecisionTreeClassifier(random_state=5805)
path = model.cost_complexity_pruning_path(opt_X_train,y_train_std)
alphas = path['ccp_alphas']
print(len(alphas))

"""test

"""


from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
import numpy as np



# Tune max_depth
max_depths = range(1, 21)
max_test_accuracy_depth = 0
optimal_max_depth = None

for max_depth in max_depths:
   model = DecisionTreeClassifier(max_depth=max_depth, random_state=5805)
   model.fit(opt_X_train, y_train_std)

   test_acc = cross_val_score(model, opt_X_test, y_test_std, cv=5).mean()

   if test_acc > max_test_accuracy_depth:
       max_test_accuracy_depth = test_acc
       optimal_max_depth = max_depth

# Tune ccp_alpha
alphas = np.linspace(0, 0.05, 20)  # Adjust the range of alphas based on your needs
max_test_accuracy_alpha = 0
optimal_alpha = None

for alpha in alphas:
   model = DecisionTreeClassifier(ccp_alpha=alpha, random_state=5805)
   model.fit(opt_X_train, y_train_std)

   # Testing accuracy using cross-validation
   test_acc = cross_val_score(model, opt_X_test, y_test_std, cv=5).mean()

   # Update optimal values if current test accuracy is higher
   if test_acc > max_test_accuracy_alpha:
       max_test_accuracy_alpha = test_acc
       optimal_alpha = alpha
```

```
# Print optimal values
print(f"Optimal max_depth: {optimal_max_depth} with max test accuracy:
{max_test_accuracy_depth:.4f}")
print(f"Optimal alpha: {optimal_alpha} with max test accuracy:
{max_test_accuracy_alpha:.4f}")
# Post-pruning Decision Tree
post_pruning_model = DecisionTreeClassifier(ccp_alpha=optimal_alpha,
random_state=5805)
post_pruning_model.fit(opt_X_train, y_train_std)

# Plot the tree
plt.figure(figsize=(20, 12))
plot_tree(post_pruning_model, rounded=True, filled=True,
feature_names=X.columns.tolist())
plt.show()

#
"""Logistic Regression"""
print("\n Logistic Regression \n")
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
precision_recall_fscore_support, auc, roc_curve
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedKFold, cross_val_score
import matplotlib.pyplot as plt
from joblib import Parallel, delayed

# Define the logistic regression model
logreg = LogisticRegression(random_state=0)

# Define the parameter grid for grid search
hyper_parameters = {
   'C': [0.01, 0.1, 1, 10],  # regularization parameter
   'penalty': ['l1', 'l2']  # regularization type
}
```

```
grid_search = GridSearchCV(logreg, hyper_parameters, cv=5, scoring='accuracy')
grid_search.fit(opt_X_train, y_train_std)
best_params = grid_search.best_params_
print(f"Best Parameters: {best_params}")
best_logreg = grid_search.best_estimator_
y_train_pred = best_logreg.predict(opt_X_train)
y_test_pred = best_logreg.predict(opt_X_test)
accuracy_train = accuracy_score(y_train_std, y_train_pred)
accuracy_test = accuracy_score(y_test_std, y_test_pred)

print(f"Training Accuracy: {accuracy_train}")
print(f"Testing Accuracy: {accuracy_test}")

# Additional evaluation metrics
print("Confusion Matrix:")
print(confusion_matrix(y_test_std, y_test_pred))

print("\nClassification Report:")
print(classification_report(y_test_std, y_test_pred))

precision, recall, fscore, support = precision_recall_fscore_support(y_test_std,
y_test_pred)

tn, fp, fn, tp = confusion_matrix(y_test_std, y_test_pred).ravel()

specificity = tn / (tn + fp)

# Print individual values
print("\nIndividual Metrics:")
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", fscore)
print("Accuracy:", accuracy_test)
print("Specificity:", specificity)

# ROC Curve and AUC
y_test_prob = best_logreg.predict_proba(opt_X_test)[:, 1]
```

```
fpr, tpr, thresholds = roc_curve(y_test_std, y_test_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC =
{:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve with AUC')
plt.legend(loc="lower right")
plt.text(0.6, 0.2, 'AUC = {:.2f}'.format(roc_auc), fontsize=12)
plt.show()

def cross_val_score_without_parallel(model, X, y, cv):
   scores = []
   X_array = X.values if isinstance(X, pd.DataFrame) else X
   y_array = y.values if isinstance(y, pd.Series) else y

   for train, test in cv.split(X_array, y_array):
       model.fit(X_array[train], y_array[train])
       y_pred = model.predict(X_array[test])
       accuracy = accuracy_score(y_array[test], y_pred)
       scores.append(accuracy)

   return scores

# Cross-validation without parallelization
n_folds = 5
stratified_kfold = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=42)
cross_val_results = cross_val_score_without_parallel(best_logreg, opt_X_train,
y_train_std, cv=stratified_kfold)

# Print cross-validation results
print("Cross-Validation Results:")
print("Individual Fold Accuracies:", cross_val_results)
print("Mean Accuracy:", sum(cross_val_results) / len(cross_val_results))
```

```
"""KNN Clustering"""
print("\n KNN clustering \n")
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
precision_recall_fscore_support, auc,
roc_curve,precision_score,recall_score,f1_score
# Choose a range of K values
k_values = range(1, 21)

# Evaluate performance for each K
accuracy_scores = []

for k in k_values:
   knn = KNeighborsClassifier(n_neighbors=k)
   knn.fit(opt_X_train, y_train_std)
   y_pred = knn.predict(opt_X_test)
   accuracy = accuracy_score(y_test_std, y_pred)
   accuracy_scores.append(accuracy)

# Plot the results
plt.plot(k_values, accuracy_scores, marker='o')
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Accuracy')
plt.title('KNN: Optimal K Value')
plt.show()

best_k = k_values[np.argmax(accuracy_scores)]
print(f"Best K value: {best_k}")

Knn_best_model = KNeighborsClassifier(n_neighbors=best_k)
Knn_best_model.fit(opt_X_train, y_train_std)
y_pred_best = Knn_best_model.predict(opt_X_test)

conf_matrix = confusion_matrix(y_test, y_pred_best)
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
```

```
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()

precision = precision_score(y_test_std, y_pred_best, average='weighted')
recall = recall_score(y_test_std, y_pred_best, average='weighted')
specificity = accuracy_score(y_test_std, y_pred_best)
f_score = f1_score(y_test_std, y_pred_best, average='weighted')

print(f"Precision: {precision:.2f}")
print(f"Recall (Sensitivity): {recall:.2f}")
print(f"Specificity: {specificity:.2f}")
print(f"F-score: {f_score:.2f}")

y_scores = Knn_best_model.predict_proba(opt_X_test)[:, 1]  # KNN does not have
predict_proba, so use decision_function
fpr, tpr, thresholds = roc_curve(y_test_std, y_scores)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='grey', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()



n_folds = 5
stratified_kfold = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=42)
cross_val_results = cross_val_score_without_parallel(Knn_best_model, opt_X_train,
y_train_std, cv=stratified_kfold)

# Print cross-validation results
print("Cross-Validation Results:")
print("Individual Fold Accuracies:", cross_val_results)
```

```
print("Mean Accuracy:", sum(cross_val_results) / len(cross_val_results))

"""SVM"""
print("\n SVM \n")
# Linear Kernel
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris

C_value = 0.01  # You can adjust this value

linear_svm = svm.SVC(kernel='linear', C=C_value)
linear_svm.fit(opt_X_train, y_train_std)

y_pred = linear_svm.predict(opt_X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Linear SVM Accuracy: {accuracy}')

conf_matrix = confusion_matrix(y_test_std, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()

precision = precision_score(y_test_std, y_pred, average='weighted')
recall = recall_score(y_test_std, y_pred, average='weighted')
specificity = accuracy_score(y_test_std, y_pred)
f_score = f1_score(y_test_std, y_pred, average='weighted')

print(f"Precision: {precision:.2f}")
print(f"Recall (Sensitivity): {recall:.2f}")
print(f"Specificity: {specificity:.2f}")
print(f"F-score: {f_score:.2f}")
```

```
# Display ROC and AUC Curve
y_scores = linear_svm.decision_function(opt_X_test)
fpr, tpr, thresholds = roc_curve(y_test_std, y_scores)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='grey', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()


n_folds = 5
stratified_kfold = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=42)
cross_val_results = cross_val_score_without_parallel(linear_svm, opt_X_train,
y_train_std, cv=stratified_kfold)

print("Cross-Validation Results:")
print("Individual Fold Accuracies:", cross_val_results)
print("Mean Accuracy:", sum(cross_val_results) / len(cross_val_results))

# Polynomial Kernel
poly_svm = svm.SVC(kernel='poly', degree=3,  C=C_value)  # You can adjust the degree
parameter
poly_svm.fit(opt_X_train, y_train_std)

# Make predictions on the test set
y_pred_poly = poly_svm.predict(opt_X_test)

# Evaluate the accuracy
accuracy_poly = accuracy_score(y_test_std, y_pred_poly)
print(f'Polynomial SVM Accuracy: {accuracy_poly}')

# Display Confusion Matrix
```

```
conf_matrix_poly = confusion_matrix(y_test_std, y_pred_poly)
sns.heatmap(conf_matrix_poly, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Polynomial SVM Confusion Matrix")
plt.show()

# Display Precision, Recall, Specificity, and F-score
precision_poly = precision_score(y_test_std, y_pred_poly, average='weighted')
recall_poly = recall_score(y_test_std, y_pred_poly, average='weighted')
specificity_poly = accuracy_score(y_test_std, y_pred_poly)
f_score_poly = f1_score(y_test_std, y_pred_poly, average='weighted')

print(f"Precision: {precision_poly:.2f}")
print(f"Recall (Sensitivity): {recall_poly:.2f}")
print(f"Specificity: {specificity_poly:.2f}")
print(f"F-score: {f_score_poly:.2f}")

# Display ROC and AUC Curve
y_scores_poly = poly_svm.decision_function(opt_X_test)
fpr_poly, tpr_poly, thresholds_poly = roc_curve(y_test_std, y_scores_poly)
roc_auc_poly = auc(fpr_poly, tpr_poly)

plt.figure(figsize=(8, 8))
plt.plot(fpr_poly, tpr_poly, label=f'AUC = {roc_auc_poly:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='grey', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Polynomial SVM ROC Curve')
plt.legend()
plt.show()



n_folds = 5
stratified_kfold = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=42)
cross_val_results = cross_val_score_without_parallel(poly_svm, opt_X_train,
y_train_std, cv=stratified_kfold)
```

```
# Print cross-validation results
print("Cross-Validation Results:")
print("Individual Fold Accuracies:", cross_val_results)
print("Mean Accuracy:", sum(cross_val_results) / len(cross_val_results))

# Radial Basis Function (RBF) Kernel
# Create an SVM model with an RBF kernel
rbf_svm = svm.SVC(kernel='rbf',  C=C_value)
rbf_svm.fit(opt_X_train, y_train_std)

# Make predictions on the test set
y_pred_rbf = rbf_svm.predict(opt_X_test)

# Evaluate the accuracy
accuracy_rbf = accuracy_score(y_test_std, y_pred_rbf)
print(f'RBF SVM Accuracy: {accuracy_rbf}')

# Display Confusion Matrix
conf_matrix_rbf = confusion_matrix(y_test_std, y_pred_rbf)
sns.heatmap(conf_matrix_rbf, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("RBF SVM Confusion Matrix")
plt.show()

# Display Precision, Recall, Specificity, and F-score
precision_rbf = precision_score(y_test_std, y_pred_rbf, average='weighted')
recall_rbf = recall_score(y_test_std, y_pred_rbf, average='weighted')
specificity_rbf = accuracy_score(y_test_std, y_pred_rbf)
f_score_rbf = f1_score(y_test_std, y_pred_rbf, average='weighted')

print(f"Precision: {precision_rbf:.2f}")
print(f"Recall (Sensitivity): {recall_rbf:.2f}")
print(f"Specificity: {specificity_rbf:.2f}")
print(f"F-score: {f_score_rbf:.2f}")
```

```
# Display ROC and AUC Curve
y_scores_rbf = rbf_svm.decision_function(opt_X_test)
fpr_rbf, tpr_rbf, thresholds_rbf = roc_curve(y_test_std, y_scores_rbf)
roc_auc_rbf = auc(fpr_rbf, tpr_rbf)

plt.figure(figsize=(8, 8))
plt.plot(fpr_rbf, tpr_rbf, label=f'AUC = {roc_auc_rbf:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='grey', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('RBF SVM ROC Curve')
plt.legend()
plt.show()


# Cross-validation without parallelization
n_folds = 5
stratified_kfold = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=42)
cross_val_results = cross_val_score_without_parallel(best_logreg, opt_X_train,
y_train_std, cv=stratified_kfold)

# Print cross-validation results
print("Cross-Validation Results:")
print("Individual Fold Accuracies:", cross_val_results)
print("Mean Accuracy:", sum(cross_val_results) / len(cross_val_results))

"""Grid Search"""
print("\n Grid Search \n")
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV, StratifiedKFold, cross_val_predict
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import (
   confusion_matrix,
   precision_score,
   recall_score,
   f1_score,
```

```
    roc_curve,
    auc
)

# Hyperparameter tuning using GridSearchCV
hyper_parameters = {
    "var_smoothing": np.logspace(0, -9, num=100),
}

nb_classifier = GaussianNB()
grid_search = GridSearchCV(
    nb_classifier, hyper_parameters, cv=StratifiedKFold(n_splits=5, shuffle=True,
random_state=42), scoring="accuracy"
)

grid_search.fit(opt_X_train, y_train_std)

# Get the best parameters from the grid search
best_params = grid_search.best_params_
print(f"Best Parameters: {best_params}")

# Use the best model from the grid search
best_classifier_Naive_bayes = grid_search.best_estimator_

# Make predictions on the test set
y_test_pred = best_classifier_Naive_bayes.predict(opt_X_test)

# Display Confusion matrix
conf_matrix = confusion_matrix(y_test_std, y_test_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Display Precision
precision = precision_score(y_test_std, y_test_pred)
print("Precision:", precision)

# Display Sensitivity or Recall
```

```
recall = recall_score(y_test_std, y_test_pred)
print("Recall (Sensitivity):", recall)

# Display Specificity
tn, fp, fn, tp = conf_matrix.ravel()
specificity = tn / (tn + fp)
print("Specificity:", specificity)

# Display F-score
f_score = f1_score(y_test_std, y_test_pred)
print("F-score:", f_score)

# Display ROC and AUC curve
y_test_prob = best_classifier_Naive_bayes.predict_proba(opt_X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test_std, y_test_prob)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color="darkorange", lw=2, label="ROC curve (AUC =
{:.2f})".format(roc_auc))
plt.plot([0, 1], [0, 1], color="navy", lw=2, linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic (ROC) Curve with AUC")
plt.legend(loc="lower right")
plt.text(0.6, 0.2, "AUC = {:.2f}".format(roc_auc), fontsize=12)
plt.show()

# Stratified K-fold cross-validation
n_folds = 5
stratified_kfold = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=42)
cross_val_results = cross_val_predict(best_classifier_Naive_bayes, opt_X_train,
y_train_std, cv=stratified_kfold, method="predict")

# Print cross-validation results
print("Stratified K-fold Cross-Validation Results:")
print("Individual Fold Predictions:", cross_val_results)
```

```
"""Random Forest Classifier

Bagging boosting stacking
"""
print("\n Random Forest \n")
# Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score



max_depth_value = 5  # You can adjust this value

# Create a Random Forest model
rf_classifier = RandomForestClassifier(n_estimators=50, max_depth=max_depth_value,
random_state=42)
rf_classifier.fit(opt_X_train, y_train_std)

# Make predictions on the test set
y_pred_rf = rf_classifier.predict(opt_X_test)

# Evaluate the accuracy
accuracy_rf = accuracy_score(y_test_std, y_pred_rf)
print(f'Random Forest Accuracy: {accuracy_rf}')

# Display Confusion Matrix
conf_matrix_rf = confusion_matrix(y_test_std, y_pred_rf)
sns.heatmap(conf_matrix_rf, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Random Forest Confusion Matrix")
plt.show()

# Display Precision, Recall, Specificity, and F-score
precision_rf = precision_score(y_test_std, y_pred_rf, average='weighted')
recall_rf = recall_score(y_test_std, y_pred_rf, average='weighted')
specificity_rf = accuracy_score(y_test_std, y_pred_rf)
```

```
f_score_rf = f1_score(y_test_std, y_pred_rf, average='weighted')

print(f"Precision: {precision_rf:.2f}")
print(f"Recall (Sensitivity): {recall_rf:.2f}")
print(f"Specificity: {specificity_rf:.2f}")
print(f"F-score: {f_score_rf:.2f}")

# Display ROC and AUC Curve
y_scores_rf = rf_classifier.predict_proba(opt_X_test)[:, 1]
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test_std, y_scores_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)

plt.figure(figsize=(8, 8))
plt.plot(fpr_rf, tpr_rf, label=f'AUC = {roc_auc_rf:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='grey', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest ROC Curve')
plt.legend()
plt.show()

# Cross-validation without parallelization
n_folds = 5
stratified_kfold = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=42)
cross_val_results = cross_val_score_without_parallel(rf_classifier, opt_X_train,
y_train_std, cv=stratified_kfold)

# Print cross-validation results
print("Cross-Validation Results:")
print("Individual Fold Accuracies:", cross_val_results)
print("Mean Accuracy:", sum(cross_val_results) / len(cross_val_results))



"""Random Forest"""
# Random Forest
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
precision_recall_fscore_support, auc, roc_curve,
precision_score,f1_score,recall_score
from sklearn.metrics import accuracy_score
#
max_depth_value = 5  # You can adjust this value

# Create a Random Forest model
rf_classifier = RandomForestClassifier(n_estimators=50, max_depth=max_depth_value,
random_state=42)
rf_classifier.fit(opt_X_train, y_train_std)

# Make predictions on the test set
y_pred_rf = rf_classifier.predict(opt_X_test)

# Evaluate the accuracy
accuracy_rf = accuracy_score(y_test_std, y_pred_rf)
print(f'Random Forest Accuracy: {accuracy_rf}')

# Display Confusion Matrix
conf_matrix_rf = confusion_matrix(y_test_std, y_pred_rf)
sns.heatmap(conf_matrix_rf, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Random Forest Confusion Matrix")
plt.show()

# Display Precision, Recall, Specificity, and F-score
precision_rf = precision_score(y_test_std, y_pred_rf, average='weighted')
recall_rf = recall_score(y_test_std, y_pred_rf, average='weighted')
specificity_rf = accuracy_score(y_test_std, y_pred_rf)
f_score_rf = f1_score(y_test_std, y_pred_rf, average='weighted')

print(f"Precision: {precision_rf:.2f}")
print(f"Recall (Sensitivity): {recall_rf:.2f}")
print(f"Specificity: {specificity_rf:.2f}")
print(f"F-score: {f_score_rf:.2f}")
```

```
# Display ROC and AUC Curve
y_scores_rf = rf_classifier.predict_proba(opt_X_test)[:, 1]
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test_std, y_scores_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)

plt.figure(figsize=(8, 8))
plt.plot(fpr_rf, tpr_rf, label=f'AUC = {roc_auc_rf:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='grey', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest ROC Curve')
plt.legend()
plt.show()

# Cross-validation without parallelization
n_folds = 5
stratified_kfold = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=42)
cross_val_results = cross_val_score_without_parallel(rf_classifier, opt_X_train,
y_train_std, cv=stratified_kfold)

# Print cross-validation results
print("Cross-Validation Results:")
print("Individual Fold Accuracies:", cross_val_results)
print("Mean Accuracy:", sum(cross_val_results) / len(cross_val_results))

"""Bagging"""
print("\n Bagging \n")
# Bagging (Bootstrap Aggregating)
from sklearn.ensemble import BaggingClassifier

# Create a BaggingClassifier with a base Random Forest model
bagging_classifier = BaggingClassifier(base_estimator=RandomForestClassifier(),
n_estimators=10, random_state=42)
bagging_classifier.fit(opt_X_train, y_train_std)

# Make predictions on the test set
```

```
predicted_y_probability_bagging = bagging_classifier.predict(opt_X_test)

# Evaluate the accuracy
accuracy_bagging = accuracy_score(y_test_std, predicted_y_probability_bagging)
print(f'Bagging Accuracy: {accuracy_bagging}')

# Display Confusion Matrix
conf_matrix_bagging = confusion_matrix(y_test_std, predicted_y_probability_bagging)
sns.heatmap(conf_matrix_bagging, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Bagging Confusion Matrix")
plt.show()

# Display Precision, Recall, Specificity, and F-score
precision_bagging = precision_score(y_test_std, predicted_y_probability_bagging,
average='weighted')
recall_bagging = recall_score(y_test_std, predicted_y_probability_bagging,
average='weighted')
specificity_bagging = accuracy_score(y_test_std, predicted_y_probability_bagging)
f_score_bagging = f1_score(y_test_std, predicted_y_probability_bagging,
average='weighted')

print(f"Precision: {precision_bagging:.2f}")
print(f"Recall (Sensitivity): {recall_bagging:.2f}")
print(f"Specificity: {specificity_bagging:.2f}")
print(f"F-score: {f_score_bagging:.2f}")

# Display ROC and AUC Curve
y_scores_bagging = bagging_classifier.predict_proba(opt_X_test)[:, 1]
fpr_bagging, tpr_bagging, thresholds_bagging = roc_curve(y_test_std,
y_scores_bagging)
roc_auc_bagging = auc(fpr_bagging, tpr_bagging)

plt.figure(figsize=(8, 8))
plt.plot(fpr_bagging, tpr_bagging, label=f'AUC = {roc_auc_bagging:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='grey', label='Random')
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Bagging ROC Curve')
plt.legend()
plt.show()



# Cross-validation without parallelization
n_folds = 5
stratified_kfold = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=42)
cross_val_results = cross_val_score_without_parallel(bagging_classifier,
opt_X_train, y_train_std, cv=stratified_kfold)

# Print cross-validation results
print("Cross-Validation Results:")
print("Individual Fold Accuracies:", cross_val_results)
print("Mean Accuracy:", sum(cross_val_results) / len(cross_val_results))



# Stacking
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression

stacking_classifier = StackingClassifier(estimators=[('rf',
RandomForestClassifier(n_estimators=100, random_state=42)),
                                                     ('dt',
DecisionTreeClassifier(random_state=42))],
                                         final_estimator=LogisticRegression())
stacking_classifier.fit(opt_X_train, y_train_std)

# Make predictions on the test set
predicted_y_probability_stacking = stacking_classifier.predict(opt_X_test)

# Evaluate the accuracy
accuracy_stacking = accuracy_score(y_test_std, predicted_y_probability_stacking)
print(f'Stacking Accuracy: {accuracy_stacking}')
```

```
# Display Confusion Matrix
conf_matrix_stacking = confusion_matrix(y_test, predicted_y_probability_stacking)
sns.heatmap(conf_matrix_stacking, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Stacking Confusion Matrix")
plt.show()

# Display Precision, Recall, Specificity, and F-score
precision_stacking = precision_score(y_test_std, predicted_y_probability_stacking,
average='weighted')
recall_stacking = recall_score(y_test_std, predicted_y_probability_stacking,
average='weighted')
specificity_stacking = accuracy_score(y_test_std, predicted_y_probability_stacking)
f_score_stacking = f1_score(y_test_std, predicted_y_probability_stacking,
average='weighted')

print(f"Precision: {precision_stacking:.2f}")
print(f"Recall (Sensitivity): {recall_stacking:.2f}")
print(f"Specificity: {specificity_stacking:.2f}")
print(f"F-score: {f_score_stacking:.2f}")

# Display ROC and AUC Curve
y_scores_stacking = stacking_classifier.predict_proba(opt_X_test)[:, 1]
fpr_stacking, tpr_stacking, thresholds_stacking = roc_curve(y_test_std,
y_scores_stacking)
roc_auc_stacking = auc(fpr_stacking, tpr_stacking)

plt.figure(figsize=(8, 8))
plt.plot(fpr_stacking, tpr_stacking, label=f'AUC = {roc_auc_stacking:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='grey', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Stacking ROC Curve')
plt.legend()
plt.show()
```

```
# Cross-validation without parallelization
n_folds = 5
stratified_kfold = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=42)
cross_val_results = cross_val_score_without_parallel(stacking_classifier,
opt_X_train, y_train_std, cv=stratified_kfold)

# Print cross-validation results
print("Cross-Validation Results:")
print("Individual Fold Accuracies:", cross_val_results)
print("Mean Accuracy:", sum(cross_val_results) / len(cross_val_results))



# Boosting (Expensive computation) Running perfectly in notebook, in python requires
a little more ram usage .
print("\n Boosting \n")
from sklearn.ensemble import AdaBoostClassifier

# Create an AdaBoost classifier with a base Random Forest model
adaboost_classifier =
AdaBoostClassifier(base_estimator=RandomForestClassifier(n_estimators=100,
random_state=42),
                                        n_estimators=50, random_state=42)
adaboost_classifier.fit(opt_X_train, y_train_std)

y_pred_adaboost = adaboost_classifier.predict(opt_X_test)

accuracy_adaboost = accuracy_score(y_test_std, y_pred_adaboost)
print(f'AdaBoost Accuracy: {accuracy_adaboost}')

# Display Confusion Matrix
conf_matrix_adaboost = confusion_matrix(y_test_std, y_pred_adaboost)
sns.heatmap(conf_matrix_adaboost, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("AdaBoost Confusion Matrix")
plt.show()
```

```
precision_adaboost = precision_score(y_test_std, y_pred_adaboost,
average='weighted')
recall_adaboost = recall_score(y_test_std, y_pred_adaboost, average='weighted')
specificity_adaboost = accuracy_score(y_test_std, y_pred_adaboost)
f_score_adaboost = f1_score(y_test_std, y_pred_adaboost, average='weighted')

print(f"Precision: {precision_adaboost:.2f}")
print(f"Recall (Sensitivity): {recall_adaboost:.2f}")
print(f"Specificity: {specificity_adaboost:.2f}")
print(f"F-score: {f_score_adaboost:.2f}")

y_scores_adaboost = adaboost_classifier.predict_proba(opt_X_test)[:, 1]
fpr_adaboost, tpr_adaboost, thresholds_adaboost = roc_curve(y_test_std,
y_scores_adaboost)
roc_auc_adaboost = auc(fpr_adaboost, tpr_adaboost)

plt.figure(figsize=(8, 8))
plt.plot(fpr_adaboost, tpr_adaboost, label=f'AUC = {roc_auc_adaboost:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='grey', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('AdaBoost ROC Curve')
plt.legend()
plt.show()


# Cross-validation without parallelization
n_folds = 5
stratified_kfold = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=42)
cross_val_results = cross_val_score_without_parallel(adaboost_classifier,
opt_X_train, y_train_std, cv=stratified_kfold)

# Print cross-validation results
print("Cross-Validation Results:")
print("Individual Fold Accuracies:", cross_val_results)
print("Mean Accuracy:", sum(cross_val_results) / len(cross_val_results))
```

```
"""Neural Networks"""
print("\n Neural Networks \n")
# Neural Network
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score

model = tf.keras.Sequential([
   tf.keras.layers.Dense(64, activation='relu',
input_shape=(opt_X_train.shape[1],)),
   tf.keras.layers.Dense(64, activation='relu'),
   tf.keras.layers.Dense(3, activation='softmax')  # Output layer with 3 neurons for
3 classes
])

model.compile(optimizer='adam',  # You can use other optimizers like 'sgd' or
'rmsprop'
              loss='sparse_categorical_crossentropy',  # For integer labels
              metrics=['accuracy'])

model.fit(opt_X_train, y_train_std, epochs=50, batch_size=32,
validation_data=(opt_X_test, y_test_std))

# Evaluate the model
predicted_y_probability = model.predict(opt_X_test)
y_pred = tf.argmax(predicted_y_probability, axis=1).numpy()
accuracy = accuracy_score(y_test_std, y_pred)
print(f'MLP Accuracy: {accuracy}')

confusion_matrix_neural_network = confusion_matrix(y_test_std, y_pred)
sns.heatmap(confusion_matrix_neural_network, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("True")
```

```
plt.title("Neural Network Confusion Matrix")
plt.show()

# Display Precision, Recall, Specificity, and F-score
precision_nn = precision_score(y_test_std, y_pred, average='weighted')
recall_nn = recall_score(y_test_std, y_pred, average='weighted')
specificity_nn = accuracy_score(y_test_std, y_pred)
f_score_nn = f1_score(y_test_std, y_pred, average='weighted')

print(f"Precision: {precision_nn:.2f}")
print(f"Recall (Sensitivity): {recall_nn:.2f}")
print(f"Specificity: {specificity_nn:.2f}")
print(f"F-score: {f_score_nn:.2f}")

# Display ROC and AUC Curve
fpr_nn, tpr_nn, thresholds_nn = roc_curve(y_test_std, predicted_y_probability[:, 1])
roc_auc_nn = auc(fpr_nn, tpr_nn)

plt.figure(figsize=(8, 8))
plt.plot(fpr_nn, tpr_nn, label=f'AUC = {roc_auc_nn:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='grey', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Neural Network ROC Curve')
plt.legend()
plt.show()



# Cross-validation without parallelization
# n_folds = 5
# stratified_kfold = StratifiedKFold(n_splits=n_folds, shuffle=True,
random_state=42)
# cross_val_results = cross_val_score_without_parallel(model, opt_X_train,
y_train_std, cv=stratified_kfold)
#
# # Print cross-validation results
# print("Cross-Validation Results:")
```

```
# print("Individual Fold Accuracies:", cross_val_results)
# print("Mean Accuracy:", sum(cross_val_results) / len(cross_val_results))


"""
**Phase 4**

---

"""

df_encoded.columns

# new_df_encoded=df_encoded.drop('salary',axis=1)
new_df_encoded=df_encoded
# Separate features (X) and target variable (y)
y = new_df_encoded['high_salary']
X = new_df_encoded.drop(columns=['high_salary'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5805)
x_scaler = StandardScaler()
X_train[['yearsExperience', 'milesFromMetropolis']] =
x_scaler.fit_transform(X_train[['yearsExperience', 'milesFromMetropolis']])
X_test[['yearsExperience', 'milesFromMetropolis']] =
x_scaler.transform(X_test[['yearsExperience', 'milesFromMetropolis']])
X_train_std = X_train
X_test_std = X_test

# Target standardization
y_scaler = StandardScaler()
y_train_std = y_scaler.fit_transform(y_train.values.reshape(-1, 1))
y_test_std = y_scaler.transform(y_test.values.reshape(-1, 1))

"""K Means clustering"""

from sklearn.metrics import silhouette_score
```

```
from sklearn.cluster import KMeans

print("------------------ Clustering ---------------------------------")

# flight_data = flight_data.drop(['price_category'], axis=1)
categorical = [var for var in X_train_std.columns if X_train_std[var].dtype == 'O']

def category_encoding(dataset):
   for var in categorical:
       ordered_labels = dataset.groupby([var])['salary'].mean().sort_values().index
       ordinal_label = {k: i for i, k in enumerate(ordered_labels, 0)}
       print(f'Encoding for {var}: {ordinal_label}')
       dataset[var] = dataset[var].map(ordinal_label)

# Applying encoding to the dataset
category_encoding(X_train_std)

scaler = StandardScaler()
scaled_data = scaler.fit_transform(X_train_std)

print("------------------ K-Mean Clustering ---------------------------------")

silhouette_scores = []
inertia_scores = []

for n_clusters in range(2, 11):
   kmeans = KMeans(n_clusters=n_clusters, random_state=0)
   cluster_labels = kmeans.fit_predict(scaled_data)
   silhouette_avg = silhouette_score(scaled_data, cluster_labels)
   silhouette_scores.append(silhouette_avg)
   inertia_scores.append(kmeans.inertia_)



plt.figure(figsize=(10, 5))
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.subplot(1, 2, 1)
```

```
plt.title('Silhouette Scores for Different Clusters')
plt.plot(range(2, 11), silhouette_scores, marker='o')


# Choose the number of clusters with the highest silhouette score
optimal_clusters = silhouette_scores.index(max(silhouette_scores)) + 2


# Plotting within-cluster variation
plt.subplot(1, 2, 2)
plt.xlabel('Number of Clusters')
plt.ylabel('Within-Cluster Variation')
plt.plot(range(2, 11), inertia_scores, marker='o')
plt.title('Within-Cluster Variation for Different Clusters')
plt.show()

"""Apriori Algorithm"""

from itertools import combinations
from collections import defaultdict

relevant_columns = [
   'yearsExperience', 'milesFromMetropolis', 'jobType_CFO',
   'jobType_JUNIOR', 'jobType_MANAGER', 'jobType_SENIOR',
   'degree_DOCTORAL', 'degree_MASTERS', 'major_BUSINESS', 'major_CHEMISTRY',
   'major_COMPSCI', 'major_ENGINEERING', 'industry_EDUCATION', 'industry_FINANCE',
'industry_OIL', 'industry_SERVICE'
]

def convert_to_transactions(data, relevant_columns):
   itemset_for_orders = []
   for _, row in data[relevant_columns].iterrows():
       transaction = [f"{col}_{row[col]}" for col in relevant_columns]
       itemset_for_orders.append(transaction)
   return itemset_for_orders

def apriori_custom(itemset_for_orders, min_support, max_length=2):
```

```
    item_count = defaultdict(int)
    num_transactions = len(itemset_for_orders)

    for transaction in itemset_for_orders:
        for item in transaction:
            item_count[frozenset([item])] += 1

    item_count = {item: count for item, count in item_count.items() if count /
num_transactions >= min_support}

    frequent_itemsets = list(item_count.keys())

    for length in range(2, max_length + 1):
        combos = combinations(frequent_itemsets, length)
        current_itemsets = defaultdict(int)

        for combo in combos:
            current_set = frozenset.union(*combo)

            if current_set not in current_itemsets:
                for transaction in itemset_for_orders:
                    if current_set.issubset(transaction):
                        current_itemsets[current_set] += 1

        current_itemsets = {item: count for item, count in current_itemsets.items()
if count / num_transactions >= min_support}
        frequent_itemsets.extend(current_itemsets.keys())

    return [(itemset, count / num_transactions) for itemset, count in
item_count.items()]

def apply_apriori_and_display(data, relevant_columns, min_support=0.01,
max_length=2, display_count=10):
    itemset_for_orders = convert_to_transactions(data, relevant_columns)
    frequent_itemsets_custom = apriori_custom(itemset_for_orders, min_support,
max_length)
```

```
    print(f"Top {display_count} frequent itemsets:")
    for itemset, support in frequent_itemsets_custom[:display_count]:
        print(f"{itemset}: {support}")

# Applying the custom Apriori algorithm on the itemset_for_orders
apply_apriori_and_display(opt_X_train, relevant_columns, min_support=0.01,
max_length=2, display_count=10)


"""DBScan"""

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt

# Perform DBSCAN clustering
dbscan = DBSCAN(eps=0.5, min_samples=50)
cluster_labels_dbscan = dbscan.fit_predict(scaled_data)

# Add the cluster labels to the original DataFrame
X_train_std['Cluster_KMeans'] = kmeans.labels_
X_train_std['Cluster_DBSCAN'] = cluster_labels_dbscan


plt.scatter(X_train_std['yearsExperience'], X_train_std['milesFromMetropolis'],
c=X_train_std['Cluster_KMeans'], cmap='viridis')
plt.title('K-Means Clustering')
plt.xlabel('yearsExperience')
plt.ylabel('milesFromMetropolis')
plt.show()

plt.scatter(X_train_std['yearsExperience'], X_train_std['milesFromMetropolis'],
c=X_train_std['Cluster_DBSCAN'], cmap='viridis')
plt.title('DBSCAN Clustering')
plt.xlabel('yearsExperience')
plt.ylabel('milesFromMetropolis')
```

```
plt.show()

from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X_train_std['yearsExperience'], X_train_std['milesFromMetropolis'],
X_train_std['salary'],
          c=X_train_std['Cluster_DBSCAN'], cmap='viridis')

ax.set_xlabel('yearsExperience')
ax.set_ylabel('milesFromMetropolis')
ax.set_zlabel('salary')

plt.title('DBSCAN Clustering (3D Scatter Plot)')
plt.show()

"""Author : Venkata Chaitanya K"""
```