- StudentID: C0902422
- Student Name: Brayan Leonardo Gil Guevara

In [1]:
```
#load the data in  C:\Users\leoko\OneDrive\Documents\DSMM\Term3\AML 3104 - Neural Networ
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import sklearn
```

Dataset Selection

The dataset that we are working with for this project, belongs to an Academic Institution looking to secure campus recruitment for its students. The objective for this project is to determinate wether the student will be recruited in campus placements.

Data Preprocessing

In [2]:
```
data =pd.read_csv('C:/Users/leoko/OneDrive/Documents/DSMM/Term3/AML 3104 - Neural Networ
data.head()
```

Out[2]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	speciali
0	1	0	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	55.0	M
1	2	0	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	86.5	M
2	3	0	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	75.0	M
3	4	0	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	66.0	M
4	5	0	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	96.8	M

What we can understand from this columns are:

- sl_no: Serial Number
- gender: Gender (0 for male, 1 for female)
- ssc_p: Secondary grades
- ssc_b: Board Secondary
- hsc_p: High School percentage
- hsc_b: Board of High School
- hsc_s: Specialization in High School
- degree_p: Degree Percentage
- degree_t: Undergraduate type
- workex: Work Experience (Yes / No)
- etest_p: E-test percentage (Employability test)
- specialisation: (Postgraduate Specialization (MBA))
- mba_p: MBA Percentage
- status: Placement Status (placed/not placed)
- salary: Salary offered (only for placed students)

'status' is our target variable. We have some categorical columns that need to be encoded 'salary' only shows a value IF the student was placed.

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 215 entries, 0 to 214
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   sl_no           215 non-null    int64  
 1   gender          215 non-null    int64  
 2   ssc_p           215 non-null    float64
 3   ssc_b           215 non-null    object 
 4   hsc_p           215 non-null    float64
 5   hsc_b           215 non-null    object 
 6   hsc_s           215 non-null    object 
 7   degree_p        215 non-null    float64
 8   degree_t        215 non-null    object 
 9   workex          215 non-null    object 
 10  etest_p         215 non-null    float64
 11  specialisation  215 non-null    object 
 12  mba_p           215 non-null    float64
 13  status          215 non-null    object 
 14  salary          148 non-null    float64
dtypes: float64(6), int64(2), object(7)
memory usage: 25.3+ KB
```

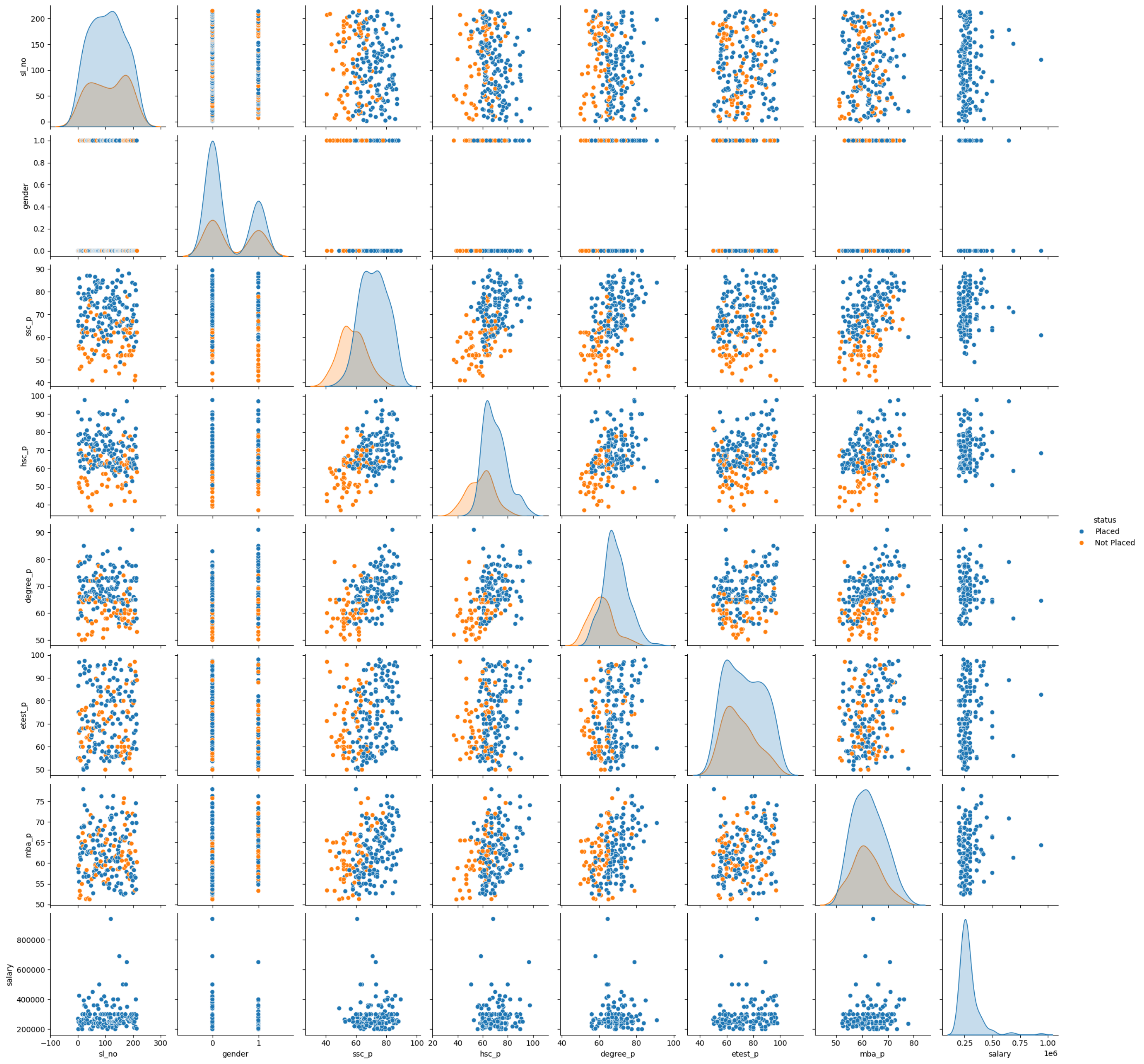
In [4]:
```
data.describe().T
```

Out[4]:

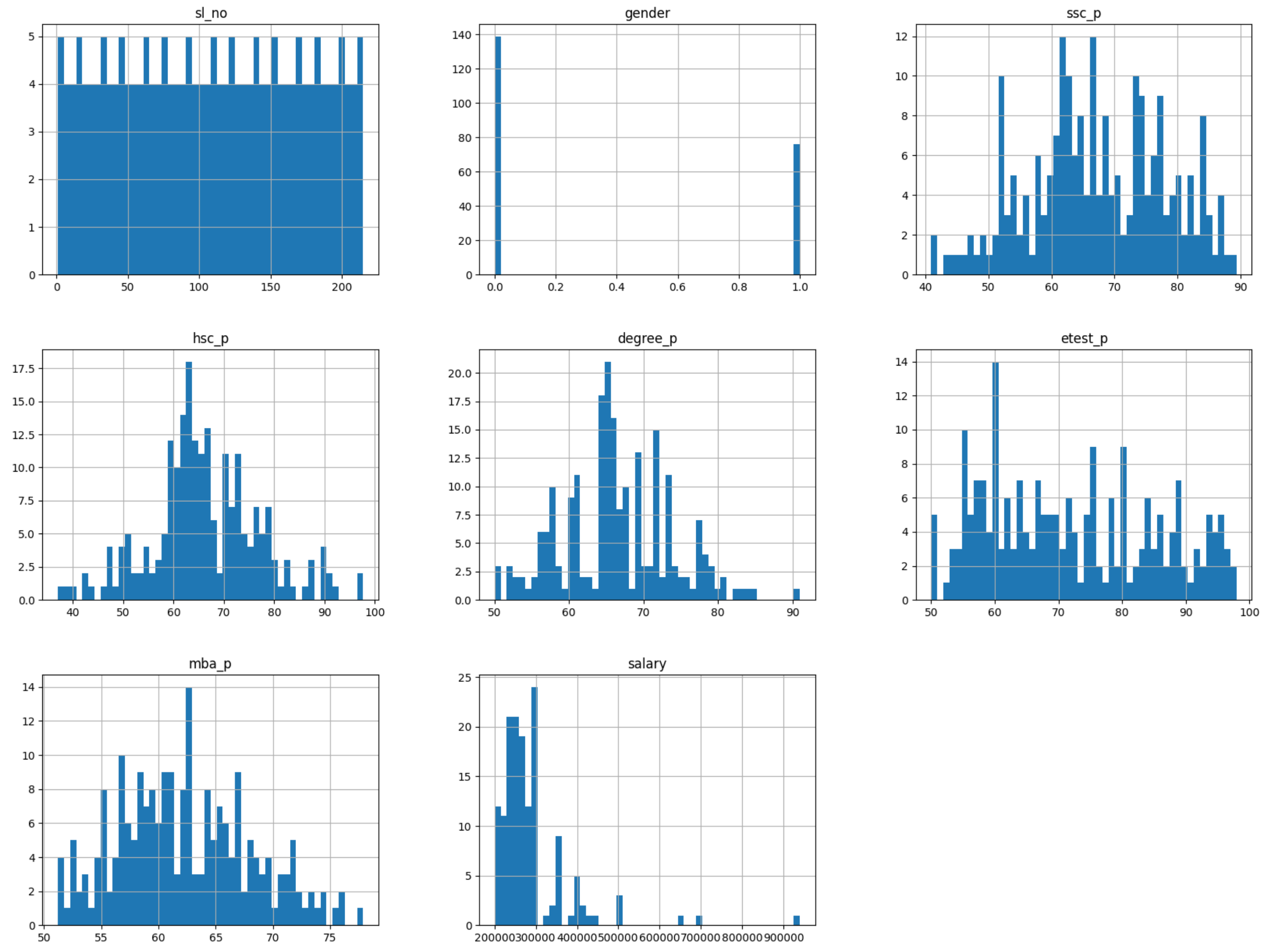
	count	mean	std	min	25%	50%	75%	max
sl_no	215.0	108.000000	62.209324	1.00	54.500	108.0	161.500	215.00
gender	215.0	0.353488	0.479168	0.00	0.000	0.0	1.000	1.00
ssc_p	215.0	67.303395	10.827205	40.89	60.600	67.0	75.700	89.40
hsc_p	215.0	66.333163	10.897509	37.00	60.900	65.0	73.000	97.70
degree_p	215.0	66.370186	7.358743	50.00	61.000	66.0	72.000	91.00
etest_p	215.0	72.100558	13.275956	50.00	60.000	71.0	83.500	98.00
mba_p	215.0	62.278186	5.833385	51.21	57.945	62.0	66.255	77.89
salary	148.0	288655.405405	93457.452420	200000.00	240000.000	265000.0	300000.000	940000.00

Let's visualize the distributions of key features

In [5]:
```
sns.pairplot(data, hue='status')
plt.show()
```



In [6]:
```
#plot density plot for the data
data.hist(bins=50, figsize=(20,15))
plt.show()
```



Handle Missing values

In [7]:
```
data.isnull().sum()
```

Out[7]:
```
sl_no              0
gender             0
ssc_p              0
ssc_b              0
hsc_p              0
hsc_b              0
hsc_s              0
degree_p           0
degree_t           0
workex             0
etest_p            0
specialisation     0
mba_p              0
status             0
salary            67
dtype: int64
```

In [8]:
```
data['salary'].fillna(0, inplace=True)
```

In [9]:
```
data.isnull().sum()
```

Out[9]:
```
sl_no             0
gender            0
ssc_p             0
ssc_b             0
hsc_p             0
```

```
hsc_b             0
hsc_s             0
degree_p          0
degree_t          0
workex            0
etest_p           0
specialisation    0
mba_p             0
status            0
salary            0
dtype: int64
```

Encode Categorial Variables

In [10]:
```
from sklearn.preprocessing import LabelEncoder
label_enc = LabelEncoder()
```

In [11]:
```
data['status'] =label_enc.fit_transform(data['status']) #This will convert the categoric
data['gender'] =label_enc.fit_transform(data[ 'gender']) #This will convert the categori
```

One-hot encode other categorial variables

In [12]:
```
cat_cols = ['ssc_b', 'hsc_b', 'hsc_s', 'degree_t', 'workex', 'specialisation'] #We defin
data = pd.get_dummies(data, columns=cat_cols, drop_first=True) #We convert the categoric
```

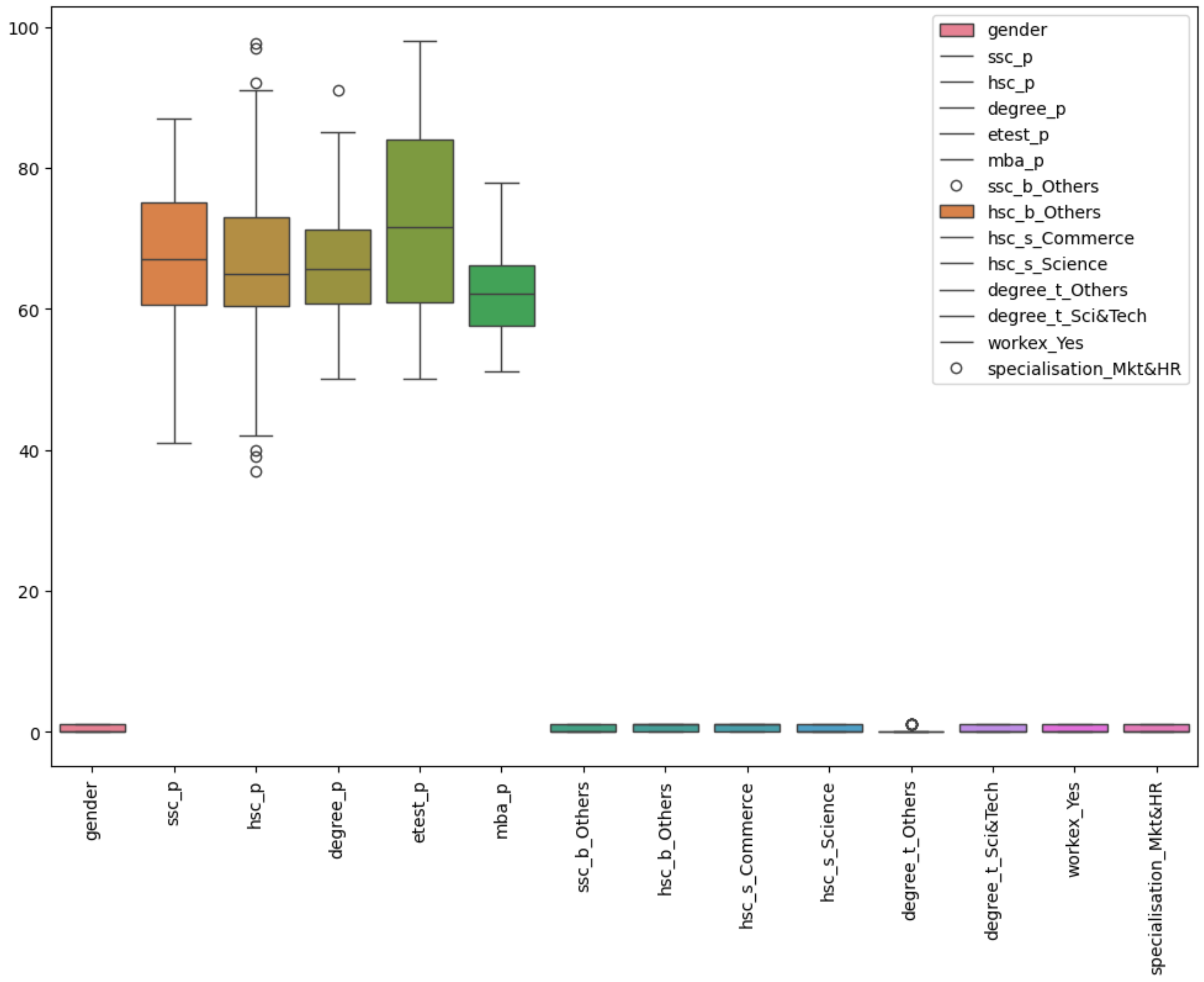
Define features and target variable

In [13]:
```
X = data.drop(['status', 'sl_no', 'salary'], axis=1)
y = data['status']
```

In [14]:
```
#Split the data into training, testing and unseen data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
```

In [15]:
```
#Lets make a boxplot to see the distribution of the data before being standardized
plt.figure(figsize=(12, 8))
sns.boxplot(data=X_train)
plt.xticks(rotation=90)
plt.legend(X.columns, loc='upper right')
```

Out[15]:
```
<matplotlib.legend.Legend at 0x220069239b0>
```

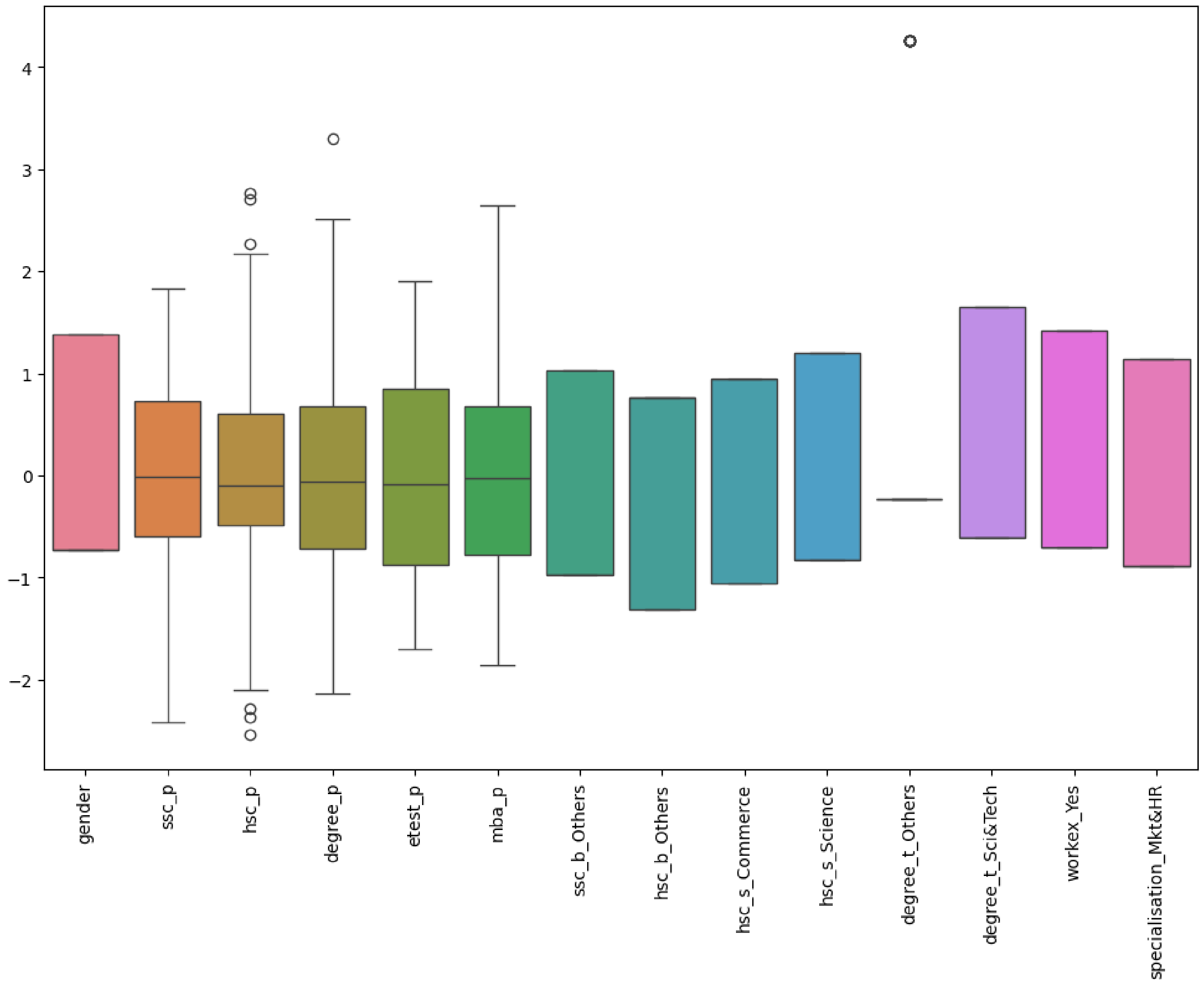


Standardize features

In [16]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [17]:
```
#Lets make a boxplot to see the distribution of the data after being standardized
plt.figure(figsize=(12, 8))
sns.boxplot(data=X_train)
plt.xticks(rotation=90)
plt.xticks(range(X_train.shape[1]), X.columns, fontsize=10, rotation=90)
```

Out[17]:
```
([<matplotlib.axis.XTick at 0x2200af4c1a0>,
  <matplotlib.axis.XTick at 0x2200d81e6f0>,
  <matplotlib.axis.XTick at 0x2200b791d60>,
  <matplotlib.axis.XTick at 0x2200d927b00>,
  <matplotlib.axis.XTick at 0x2200d891010>,
  <matplotlib.axis.XTick at 0x2200d959d90>,
  <matplotlib.axis.XTick at 0x2200d95a600>,
  <matplotlib.axis.XTick at 0x2200d82c650>,
  <matplotlib.axis.XTick at 0x2200d95ad80>,
  <matplotlib.axis.XTick at 0x2200d95b5c0>,
  <matplotlib.axis.XTick at 0x2200d95bf50>,
  <matplotlib.axis.XTick at 0x2200d9808c0>,
  <matplotlib.axis.XTick at 0x2200d8085f0>,
  <matplotlib.axis.XTick at 0x2200d95a990>],
 [Text(0, 0, 'gender'),
  Text(1, 0, 'ssc_p'),
  Text(2, 0, 'hsc_p'),
  Text(3, 0, 'degree_p'),
  Text(4, 0, 'etest_p'),
  Text(5, 0, 'mba_p'),
  Text(6, 0, 'ssc_b_Others'),
  Text(7, 0, 'hsc_b_Others'),
  Text(8, 0, 'hsc_s_Commerce'),
  Text(9, 0, 'hsc_s_Science'),
  Text(10, 0, 'degree_t_Others'),
  Text(11, 0, 'degree_t_Sci&Tech'),
  Text(12, 0, 'workex_Yes'),
  Text(13, 0, 'specialisation_Mkt&HR')])
```



Model Selection

In [18]:
```python
#For the Classification problem ahead we will be using three different models, Logistic
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
```

In [19]:
```python
#Lets start by initializing all our models
log_reg = LogisticRegression()
rf = RandomForestClassifier()
gbc = GradientBoostingClassifier()
svc = SVC()
```

In [20]:
```
#Now, let's do some hyperparameter tuning for each model
from sklearn.model_selection import GridSearchCV

#Logistic Regression
log_reg_params = {'penalty': ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
grid_log_reg = GridSearchCV(log_reg, log_reg_params)
grid_log_reg.fit(X_train, y_train)
log_reg = grid_log_reg.best_estimator_

#Random Forest
rf_params = {'n_estimators': [100, 200, 500], 'max_depth': [3, 5, 7]}
grid_rf = GridSearchCV(rf, rf_params)
grid_rf.fit(X_train, y_train)
rf = grid_rf.best_estimator_

#Gradient Boosting
gbc_params = {'n_estimators': [100, 200, 500], 'learning_rate': [0.05, 0.1, 0.5], 'max_d
grid_gbc = GridSearchCV(gbc, gbc_params)
grid_gbc.fit(X_train, y_train)
gbc = grid_gbc.best_estimator_



#SVC
svc_params = {'C': [0.1, 1, 10], 'gamma': [0.1, 0.01, 0.001]}
grid_svc = GridSearchCV(svc, svc_params)
grid_svc.fit(X_train, y_train)
svc = grid_svc.best_estimator_
```

```
C:\Users\leoko\AppData\Roaming\Python\Python312\site-packages\sklearn\model_selection\_v
alidation.py:425: FitFailedWarning:
35 fits failed out of a total of 70.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='ra
ise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
35 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\leoko\AppData\Roaming\Python\Python312\site-packages\sklearn\model_sele
ction\_validation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\leoko\AppData\Roaming\Python\Python312\site-packages\sklearn\base.py",
line 1152, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\leoko\AppData\Roaming\Python\Python312\site-packages\sklearn\linear_mod
el\_logistic.py", line 1169, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\leoko\AppData\Roaming\Python\Python312\site-packages\sklearn\linear_mod
el\_logistic.py", line 56, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\leoko\AppData\Roaming\Python\Python312\site-packages\sklearn\model_selection\_s
earch.py:979: UserWarning: One or more of the test scores are non-finite: [       nan 0.
68016807        nan 0.80218487        nan 0.86655462
        nan 0.87210084        nan 0.85462185        nan 0.8487395
        nan 0.84285714]
  warnings.warn(
```

Model Training

In [21]:
```python
#Now, let's train each model
log_reg.fit(X_train, y_train)
rf.fit(X_train, y_train)
gbc.fit(X_train, y_train)
svc.fit(X_train, y_train)
```

Out[21]:
```
SVC
SVC(C=10, gamma=0.01)
```

Model Evaluation

In [22]:
```python
#To evaluate the models, we will use the accuracy score, the precision, recall, f1-score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, con

models = [log_reg, rf, gbc, svc]
model_names = ['Logistic Regression', 'Random Forest', 'Gradient Boosting', 'SVC']
```

Voting Classifier

In [23]:
```python
#Implement voting classifier
from sklearn.ensemble import VotingClassifier

voting_clf = VotingClassifier(estimators=[('lr', log_reg), ('rf', rf), ('gbc', gbc), ('s
voting_clf
```

Out[23]:
```
                              VotingClassifier
          lr                      rf                              gbc                  svc
 LogisticRegression   RandomForestClassifier   GradientBoostingClassifier   SVC
```

In [24]:
```python
voting_clf.fit(X_train, y_train)
```

Out[24]:
```
                              VotingClassifier
          lr                      rf                              gbc                  svc
 LogisticRegression   RandomForestClassifier   GradientBoostingClassifier   SVC
```

In [25]:
```python
vote_pred = voting_clf.predict(X_test)
voting_clf.score(X_test, y_test)
voting_accuracy = accuracy_score(y_test, vote_pred)
voting_report = classification_report(y_test, vote_pred)
```

In [26]:
```python
#Metrics time, we wull mesuare the avg value for the metrics
print('Voting Classifier')
print('Accuracy: ', voting_accuracy)
print(voting_report)
print('\n')
```

```
Voting Classifier
Accuracy:  0.8604651162790697
              precision    recall  f1-score   support
```

```
           0       0.80      0.67      0.73        12
           1       0.88      0.94      0.91        31

    accuracy                           0.86        43
   macro avg       0.84      0.80      0.82        43
weighted avg       0.86      0.86      0.86        43
```

In [27]:
```
#Let's see individual score per model
for i, model in enumerate(models):
    y_pred = model.predict(X_test)
    print(model_names[i])
    print('Accuracy: ', accuracy_score(y_test, y_pred))
    print('Precision: ', precision_score(y_test, y_pred))
    print('Recall: ', recall_score(y_test, y_pred))
    print('F1 Score: ', f1_score(y_test, y_pred))
    print(confusion_matrix(y_test, y_pred))
    print('\n')
```

```
Logistic Regression
Accuracy:  0.8837209302325582
Precision:  0.90625
Recall:  0.9354838709677419
F1 Score:  0.9206349206349206
[[ 9  3]
 [ 2 29]]


Random Forest
Accuracy:  0.7906976744186046
Precision:  0.8055555555555556
Recall:  0.9354838709677419
F1 Score:  0.8656716417910448
[[ 5  7]
 [ 2 29]]


Gradient Boosting
Accuracy:  0.8372093023255814
Precision:  0.875
Recall:  0.9032258064516129
F1 Score:  0.8888888888888888
[[ 8  4]
 [ 3 28]]


SVC
Accuracy:  0.8604651162790697
Precision:  0.8787878787878788
Recall:  0.9354838709677419
F1 Score:  0.90625
[[ 8  4]
 [ 2 29]]
```

Report

1. Dataset Description:

It contains various features about students, including their academic performance, demographics, and placement status. The key columns in the dataset include:

- sl_no: Serial number
- gender: Gender of the student
- ssc_p: Secondary Education percentage (10th Grade)
- ssc_b: Board of Education for SSC (10th Grade) - Central/ Others
- hsc_p: Higher Secondary Education percentage (12th Grade)
- hsc_b: Board of Education for HSC (12th Grade)
- hsc_s: Specialization in Higher Secondary Education
- degree_p: Degree Percentage
- degree_t: Type of undergraduate degree
- workex: Work Experience
- etest_p: E-test percentage
- specialisation: Postgraduate Specialization
- mba_p: MBA percentage
- status: Placement status (Placed/Not Placed)
- salary: Salary offered (if placed)

1. Preprocessing Steps

   - Data Loading: The dataset is loaded into a pandas DataFrame.
     - Exploratory Data Analysis (EDA):
     - Used head(), info(), and describe().T to understand the dataset structure and summary statistics.
     - Visualized data distributions using pair plots and histograms.
     - Handling Missing Values:
     - Checked for missing values using isnull().sum().
     - Filled missing values in the salary column with 0.
     - Encoding Categorical Variables:
     - Used LabelEncoder to convert binary categorical variables (status and gender) into numerical format.
     - Applied one-hot encoding to multi-class categorical variables (ssc_b, hsc_b, hsc_s, degree_t, workex, and specialisation).
     - Feature Selection:
     - Dropped irrelevant columns (sl_no and salary) and the target column (status).
     - Separated features (X) and target variable (y).
     - Data Splitting:
     - Split the dataset into training (80%) and testing (20%) sets using train_test_split.
     - Feature Scaling:
     - Applied StandardScaler to standardize the feature values.
2. Model Selection and Explanation Models Chosen

- Logistic Regression:

A simple and effective linear model for binary classification. Suitable as a baseline model due to its interpretability and efficiency.

- Random Forest:

An ensemble method using multiple decision trees to improve performance. Chosen for its robustness to overfitting and ability to handle both numerical and categorical data.

- Gradient Boosting Classifier (GBC):

An advanced ensemble technique that builds trees sequentially, each trying to correct the errors of the previous one. Selected for its high accuracy and ability to handle complex data patterns.

- Support Vector Classifier (SVC):

A powerful classifier that finds the optimal hyperplane to separate classes. Chosen for its effectiveness in high-dimensional spaces and versatility with different kernel functions.

- Voting Classifier:

An ensemble method that combines the predictions of the aforementioned models using majority voting. Selected to leverage the strengths of individual models for improved performance.

1. Model Evaluation

Evaluation Metrics

- Accuracy: The proportion of correctly predicted instances.
- Precision: The ratio of true positive predictions to the total predicted positives.
- Recall: The ratio of true positive predictions to the total actual positives.
- F1 Score: The harmonic mean of precision and recall.
- Confusion Matrix: A summary of prediction results on the classification problem.

Model Performances

- Logistic Regression:

```
Accuracy:  0.8837209302325582
Precision:  0.90625
Recall:  0.9354838709677419
F1 Score:  0.9206349206349206
[[ 9  3]
 [ 2 29]]
```

- Random Forest:

```
Accuracy:  0.7906976744186046
Precision:  0.8055555555555556
Recall:  0.9354838709677419
F1 Score:  0.8656716417910448
[[ 5  7]
 [ 2 29]]
```

- Gradient Boosting Classifier (GBC):

```
Accuracy:  0.8372093023255814
Precision:  0.875
Recall:  0.9032258064516129
F1 Score:  0.8888888888888888
[[ 8  4]
 [ 3 28]]
```

- Support Vector Classifier (SVC):

```
Accuracy:  0.8604651162790697
Precision:  0.8787878787878788
Recall:  0.9354838709677419
F1 Score:  0.90625
[[ 8  4]
[ 2 29]]
```

- Voting Classifier:

Accuracy: 0.8604651162790697 precision recall f1-score support

```
          0       0.80      0.67      0.73        12
          1       0.88      0.94      0.91        31

   accuracy                           0.86        43
```

macro avg 0.84 0.80 0.82 43 weighted avg 0.86 0.86 0.86 43

Best Performing Model

The results obtained support the decision of chosing the LogRegression as the best performer for this short project. The reason behind it can be understood after noticing:

- It contains the higher accuracy among all other options. 88.37%
- Higher F1 score. It also has the highest score at 92.06%, this is a balanced measure considering both precision and recall.
- High precision and recall. This indicates that it performs well in idenitifying both the positives and negatives correctly.

  On a side note, the ensemble approach of the Voting Classifier also performed well, indicating that combining multiple models can result in robust performance. However, it did not surpass the performance of the Logistic Regression in terms of the F1 score.