SFBU **Name:** _____
Applied Machine Learning
Final Review Package

---

1. I will ask either the exact or the slightly modified version of the following problems in the final.
2. I may ask interpretations of any result or comparisons based on these questions.
3. You will have 5 problems in the final.
4. Since you have the questions, I expect everyone behave professionally during the test.

---

1. You are given a dataset with missing values in multiple columns. Implement a strategy to handle missing data using appropriate techniques

Solution:

```
import pandas as pd
from sklearn.impute import SimpleImputer

# Assuming 'df' is your DataFrame
imputer = SimpleImputer(strategy='mean')
df_imputed = pd.DataFrame(imputer.fit_transform(df),
columns=df.columns)
```

2. Explain the concept of feature scaling. Apply Min-Max scaling and standard scale normalization to a given dataset. Discuss the impact on model performance.

Solution:

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Min-Max Scaling
scaler_minmax = MinMaxScaler()
df_minmax = pd.DataFrame(scaler_minmax.fit_transform(df),
columns=df.columns)

# Standard Normalization
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
```

3. Categorical Data Encoding. Given a dataset with categorical variables, perform one-hot encoding or label encoding. Discuss the pros and cons of each method.

Solution:

```
# One-Hot Encoding
df_encoded = pd.get_dummies(df, columns=['categorical_column'],\
drop_first=True)

# Label Encoding
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df['categorical_column'] = label_encoder.\
fit_transform(df['categorical_column'])
```

4. Simple Linear Regression. Implement simple linear regression on a dataset and interpret the coefficients. Visualize the regression line along with the scatter plot.

Solution:

```
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

model = LinearRegression()
X = df[['feature']]
y = df['target']
model.fit(X, y)

plt.scatter(X, y)
plt.plot(X, model.predict(X), color='red')
```

5. Binary Classification. Apply logistic regression to a binary classification problem. Evaluate the model using precision, recall, and F1-score. Discuss the ROC curve.

Solution:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,\
roc_curve, roc_auc_score
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)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
fpr, tpr, thresholds = roc_curve(y_test,\
model.predict_proba(X_test)[:, 1])
auc = roc_auc_score(y_test, y_pred)
```

6. SVM for Classification. Implement SVM for a binary classification problem. Experiment with different kernel functions (e.g., linear, polynomial, radial basis function) and discuss their impact.

Solution:

```
from sklearn.svm import SVC

# Try kernel='poly' or kernel='rbf' for different kernels
model_svm = SVC(kernel='linear')

model_svm.fit(X_train, y_train)
```

7. Decision Trees. Visualize a decision tree for a given dataset. Discuss how the tree makes decisions and its interpretability.

Solution:

```
from sklearn.tree import DecisionTreeClassifier, plot_tree

model_tree = DecisionTreeClassifier()
model_tree.fit(X, y)

plt.figure(figsize=(10, 7))
plot_tree(model_tree, filled=True,\
feature_names=df.columns[:-1],\
class_names=['class_0', 'class_1'])
plt.show()
```

8. Random Forest Classifier. Implement a random forest classifier and analyze the feature importance. Compare the results with a single decision tree.

Solution:

```
from sklearn.ensemble import RandomForestClassifier

model_rf = RandomForestClassifier()
model_rf.fit(X_train, y_train)
feature_importances = model_rf.feature_importances_
```

9. KNN for Classifier. Apply KNN to a classification problem. Experiment with different values of k and discuss their impact on model performance.

Solution:

```
from sklearn.neighbors import KNeighborsClassifier

model_knn = KNeighborsClassifier(n_neighbors=3)
# Try different values for 'n_neighbors'
model_knn.fit(X_train, y_train)
```

10. Cross Validation. Implement k-fold cross-validation to assess model performance. Discuss how cross-validation helps in obtaining a more reliable estimate of model performance.

Solution:

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X, y, cv=5)  # 5-fold cross-validation
```

11. Hyperparameter Tuning. Perform hyperparameter tuning for one of the models (e.g., SVM, Random Forest) using grid search or random search. Discuss the impact of hyperparameter choices on model performance.

Solution:

```
from sklearn.model_selection import GridSearchCV

param_grid = {'param1': [value1, value2], 'param2': [value3, value4]}
grid_search = GridSearchCV(model, param_grid, cv=3)
grid_search.fit(X, y)
best_params = grid_search.best_params_
```

12. Data Preparation for Classification. Given a dataset containing both numerical and categorical features, perform the necessary preprocessing steps, including handling missing data, feature scaling, and encoding categorical variables. Choose appropriate preprocessing techniques based on the nature of the features.

Solution:

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Assuming 'df' is your DataFrame with both numerical and categorical features

# Preprocessing pipeline
numeric_features = df.select_dtypes(include=['int64', 'float64']).columns
categorical_features = df.select_dtypes(include=['object']).columns

numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder())
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Applying the preprocessing
X = preprocessor.fit_transform(df)
```

13. Logistic Regression with Preprocessed Data. Apply logistic regression to a dataset (X, and y) for a binary classification task. Assume that X is already preprocessed. Evaluate the model using relevant metrics and visualize the results, such as a confusion matrix and ROC curve. Discuss how preprocessing impacted the model's performance.

Solution:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, roc_auc_score

# Assuming 'X' is the preprocessed feature matrix and
#'y' is the target variable

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

# Logistic Regression
model_lr = LogisticRegression()
model_lr.fit(X_train, y_train)

# Evaluate the model
y_pred = model_lr.predict(X_test)
print(classification_report(y_test, y_pred))

# Visualize ROC curve
roc_auc = roc_auc_score(y_test, model_lr.predict_proba(X_test)[:, 1])
```

14. Support Vector Machines (SVM) with Feature Engineering. Performing feature engineering, such as creating interaction terms or polynomial features. Implement an SVM classifier on the modified dataset and experiment with different kernel functions.

Solution:

```
from sklearn.svm import SVC
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

# Feature Engineering
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

# SVM Classifier
model_svm = SVC(kernel='rbf')  # Try 'poly' or 'rbf' for different kernels
model_svm.fit(X_poly, y)
```

15. Model Evaluation and Comparison. Given a dataset, preprocess the data and implement multiple classification algorithms (e.g., Logistic Regression, Random Forest, SVM, KNN). Use cross-validation for model evaluation and compare the performance metrics.

Solution:

```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

# Assuming 'X' is the preprocessed feature matrix and
#'y' is the target variable

# Models
models = [
    ('Random Forest', RandomForestClassifier()),
    ('Logistic Regression', LogisticRegression()),
    ('SVM', SVC()),
    ('KNN', KNeighborsClassifier())
]

# Model Evaluation and Comparison
for name, model in models:
    scores = cross_val_score(model, X, y, cv=5)
    print(f"{name}: {scores.mean()} (Accuracy)")
```

16. Decision Trees for Regression. Apply decision trees to a regression problem. Preprocess the data accordingly and evaluate the performance of the decision tree model in predicting a continuous target variable. Discuss the interpretability of the decision tree in a regression context.

Solution:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Assuming 'X' is the preprocessed feature matrix and
#'y' is the target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,\
test_size=0.2, random_state=42)

# Decision Tree Regression
model_dt_reg = DecisionTreeRegressor()
model_dt_reg.fit(X_train, y_train)

# Make predictions
y_pred = model_dt_reg.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Visualize the decision tree (simplified)
plt.figure(figsize=(10, 7))
plot_tree(model_dt_reg, filled=True, feature_names=X.columns);
```

17. Random Forest Classifier. Utilize a random forest classifier on a dataset with both numerical and categorical features. Extract and analyze feature importance from the random forest model. Discuss the advantages of using a random forest for feature selection.

Solution:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
# Assuming 'df' is your DataFrame with features and target variable
# Separate features and target variable
X = df.drop('target', axis=1)
y = df['target']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,\
test_size=0.2, random_state=42)

# Create a random forest classifier and fit
model_rf = RandomForestClassifier(n_estimators=100, random_state=42)
model_rf.fit(X_train, y_train)

# Extract feature importances
feature_importances = model_rf.feature_importances_

# Create a DataFrame to display feature importance
feature_importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances
})

# Sort the DataFrame by importance in descending order
feature_importance_df = feature_importance_df.\
sort_values(by='Importance', ascending=False)

# Display feature importance
print(feature_importance_df)
```

```
# Visualize feature importance
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'], \
feature_importance_df['Importance'])
plt.xlabel('Importance')
plt.title('Feature Importance from Random Forest')
```

18. Data Manipulation. Consider a dataset containing information about students' exam scores, subjects, and other attributes. The dataset is stored in a Pandas DataFrame named student_data.

    Slicing and Filtering:

    Task 1.1: Slice the DataFrame to select only the rows corresponding to students who scored above 80 in the 'Math' subject.

    Task 1.2: Filter the DataFrame to include only the columns 'StudentID', 'Math', and 'Science'.

    Grouping:

    Task 2.1: Group the DataFrame by the 'Grade' column and calculate the mean score for each subject within each grade.

    Task 2.2: Find the grade with the highest average score in 'English'.

    Selecting Rows and Columns:

    Task 3.1: Select the row where the 'StudentID' is 101.

    Task 3.2: Select all rows where the 'Math' score is below 70 and the 'Science' score is above 75.

Solution:

```
import pandas as pd

# Assuming 'student_data' is your DataFrame

# Task 1.1:
above_80_math = student_data[student_data['Math'] > 80]

# Task 1.2:
selected_columns = student_data[['StudentID', 'Math', 'Science']]

# Task 2.1:
mean_scores_by_grade = student_data.groupby('Grade').mean()

# Task 2.2:
highest_avg_english_grade = mean_scores_by_grade['English'].idxmax()

# Task 3.1:
row_student_id_101 = student_data[student_data['StudentID'] == 101]

# Task 3.2:
filtered_rows = student_data[(student_data['Math'] < 70) & \
(student_data['Science'] > 75)]
```

19. Data Manipulation. Consider a dataset containing information about employees, including their department, salary, and performance ratings. The dataset is stored in a Pandas DataFrame named employee_data.

    Slicing and Filtering:

    Task 1.1: Slice the DataFrame to select only the rows corresponding to employees in the 'Engineering' department.

    Task 1.2: Filter the DataFrame to include only the columns 'EmployeeID', 'Salary', and 'PerformanceRating'.

    Grouping:

    Task 2.1: Group the DataFrame by the 'Department' column and calculate the median salary for each department. Task 2.2: Find the department with the highest median performance rating.

    Selecting Rows and Columns:

    Task 3.1: Select the rows where the 'Salary' is above $80,000. Task 3.2: Select all rows where the 'Department' is 'Sales' and the 'PerformanceRating' is 4.

Solution:

```
import pandas as pd

# Assuming 'employee_data' is your DataFrame

# Task 1.1:
engineering_department = employee_data[employee_data['Department']\
== 'Engineering']


# Task 1.2:
selected_columns_employee = employee_data[['EmployeeID',\
'Salary', 'PerformanceRating']]

# Task 2.1:
median_salary_by_department = employee_data.groupby('Department')\
['Salary'].median()

# Task 2.2:
highest_median_rating_department = median_salary_by_department.idxmax()

# Task 3.1:
high_salary_employees = employee_data[employee_data['Salary'] > 80000]

# Task 3.2:
sales_department_rating_4 = employee_data[(employee_data['Department'] == \
'Sales') & (employee_data['PerformanceRating'] == 4)]
```

20. Data Manipulation. Consider a dataset containing information about customer orders, including order IDs, product names, order dates, and quantities. The dataset is stored in a Pandas DataFrame named order_details.

    Parsing and Changing Data Types:

    Task 1.1: Parse the 'OrderDate' column from a string to a datetime object.

    Task 1.2: Change the data type of the 'Quantity' column from integer to string.

    Task 1.3: Create a new column named 'Year' by extracting the year from the 'OrderDate' using a lambda function.

    Displaying Results:

    Task 2.1: Display the first 5 rows of the DataFrame after completing the data type changes.

    Task 2.2: Provide summary statistics for the 'Quantity' column after the data type change.

Solution:

```
import pandas as pd

# Assuming 'order_details' is your DataFrame

# Task 1.1:
order_details['OrderDate'] = pd.to_datetime(order_details['OrderDate'])

# Task 1.2:
order_details['Quantity'] = order_details['Quantity'].astype(str)

# Task 1.3:
order_details['Year'] = order_details['OrderDate'].\
apply(lambda x: x.year)

# Task 2.1:
print("Task 2.1: First 5 rows after data type changes")
print(order_details.head())

# Task 2.2:
print("\nTask 2.2: Summary statistics for 'Quantity' \
after data type change")
print(order_details['Quantity'].describe())
```

21. Model Evaluation. You are working on a classification project. The classification task involves predicting whether a customer will subscribe to a service ('Yes' or 'No'). Below are the predictions and actual values:

    Classification (Subscription Prediction):

    Model Predictions: ['No', 'Yes', 'No', 'Yes', 'Yes']

    Actual Subscriptions: ['Yes', 'No', 'Yes', 'Yes', 'No']

    Classification Evaluation Metrics:

    Task 2.1: Calculate precision, recall, and F1-score for the Model.

    Task 2.2: Considering precision, recall, and F1-score for the Model, assess its overall classification performance. Provide insights into the model's strengths and weaknesses.

Solution: