This process will include data loading, cleaning, visualization, and model creation.

---

1. Define the Problem and Objective

- **Objective**: Predict player performance or salaries based on various features such as points per game, assists, rebounds, field goal percentage, and others.
- **Goal**: Perform EDA to understand the relationships between features and the target variable (e.g., `salary`, `points`, `win shares`), and build a model to predict player performance or salary.

---

2. Load the Dataset

```
import pandas as pd

# Load NBA Players dataset
df = pd.read_csv('nba_players.csv')

# Display the first few rows of the dataset
df.head()
```

---

3. Data Exploration (EDA)

Check the Shape and Data Types

```
# Check dataset shape
print(f"Dataset Shape: {df.shape}")

# Data types and missing values
print(df.info())

# Summary statistics for numerical columns
print(df.describe())
```

Visualize Distributions

```
import matplotlib.pyplot as plt
import seaborn as sns

# Distribution of Points per Game (PTS)
sns.histplot(df['PTS'], kde=True)
plt.title("Points per Game Distribution")
plt.show()

# Distribution of Salaries
sns.histplot(df['SALARY'], kde=True)
plt.title("Salary Distribution")
plt.show()

# Correlation heatmap
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Heatmap")
plt.show()
```

---

4. Data Cleaning

Handle Missing Values

```
# Check for missing values
print(df.isnull().sum())

# Handle missing values
# Example: Fill missing salary values with the median salary
df['SALARY'].fillna(df['SALARY'].median(), inplace=True)

# Check if missing values are handled
print(df.isnull().sum())
```

Handle Duplicates

```
# Remove duplicate rows if any
df.drop_duplicates(inplace=True)
```

---

5. Feature Engineering (Optional)

Create New Features (if needed)

```
# Example: Create a new feature 'age_group' to categorize players by age
df['age_group'] = df['AGE'].apply(lambda x: 'Young' if x <= 25 else ('Prime'
if x <= 30 else 'Veteran'))

# Display first few rows to check new feature
print(df[['AGE', 'age_group']].head())
```

---

6. Data Preprocessing

Convert Categorical Variables to Numerical

```
# Convert 'age_group' into numerical values using mapping
df['age_group'] = df['age_group'].map({'Young': 0, 'Prime': 1, 'Veteran': 2})

# Check the data after transformation
print(df[['age_group']].head())
```

Scale Numerical Features

```
from sklearn.preprocessing import StandardScaler

# Initialize the scaler
scaler = StandardScaler()

# Scale the numerical features (e.g., points per game, salary, etc.)
scaled_features = df[['PTS', 'AST', 'REB', 'FG%', 'SALARY']]  # Specify
numerical columns to scale
df[['PTS', 'AST', 'REB', 'FG%', 'SALARY']] =
scaler.fit_transform(scaled_features)
```

Split the Data into Features (X) and Target (y)

```
# Define features (X) and target (y)
X = df.drop(columns=['SALARY'])  # Exclude target variable (e.g., 'SALARY')
y = df['SALARY']  # or you could choose another target such as 'PTS',
'WIN_SHARE', etc.
```

---

7. Split the Data into Training and Testing Sets

```
from sklearn.model_selection import train_test_split

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

---

8. Model Selection

Choose the Model

```
from sklearn.ensemble import RandomForestRegressor

# Initialize the Random Forest model
model = RandomForestRegressor(n_estimators=100, random_state=42)
```

---

9. Model Training

```
# Train the model on the training data
model.fit(X_train, y_train)
```

---

10. Model Evaluation

Predictions and Evaluation Metrics

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Make predictions
```

```
y_pred = model.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Absolute Error: {mae}')
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

Residuals Plot

```
# Plot residuals to understand prediction errors
sns.scatterplot(x=y_test, y=y_test - y_pred)
plt.title("Residuals Plot")
plt.xlabel("True Values")
plt.ylabel("Residuals")
plt.show()
```

---

11. Model Interpretation and Insights

Feature Importance

```
# Get feature importances from the trained model
importances = model.feature_importances_

# Create a DataFrame to display features and their importance
features = X.columns
feature_importance_df = pd.DataFrame({'Feature': features, 'Importance':
importances})

# Sort by importance
feature_importance_df = feature_importance_df.sort_values(by='Importance',
ascending=False)

# Plot feature importances
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title("Feature Importance")
plt.show()

print(feature_importance_df)
```

---

12. Fine-Tuning and Optimization (Optional)

Hyperparameter Tuning

```
from sklearn.model_selection import GridSearchCV

# Set up grid search parameters for Random Forest Regressor
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=3,
n_jobs=-1, verbose=2)

# Fit grid search
grid_search.fit(X_train, y_train)

# Get best parameters
print("Best Parameters:", grid_search.best_params_)
```

---

13. Final Reporting

Summarize Insights

- **EDA Insights**: Discuss the distribution of features like points per game, assists, and rebounds. Explore correlations between player stats and salary or performance metrics like win shares.
- **Model Evaluation**: Summarize the Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared values for the model's performance.
- **Important Features**: Identify which features most affect the target variable (e.g., salary, performance).
- **Next Steps**: Try different models (e.g., XGBoost, linear regression) or perform hyperparameter tuning to improve results.

Additional Notes:

- Start with a **baseline model** like Random Forest Regressor to predict performance or salary.
- **Cross-validation** can be used for more robust model performance evaluation.
- Explore **feature engineering** by creating new features like performance ratios or career averages.
- **Hyperparameter tuning** could further improve model performance for better predictions.