Main 7

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

#Data Sets
data = pd.read_csv("C:/Users/Administrator/Downloads/player_data.csv")
new_data = pd.read_csv("extended_player_data.csv")

combined_data = pd.concat([data, new_data], ignore_index=True)
new_data = pd.read_csv("extended_player_data.csv")

# Separating attribute and target columns
X = data[["Finishing", "Pass", "Technique", "Speed", "Strength",
"Stamina"]]
y = data["Transfer Value"]
X_new = new_data[["Finishing", "Pass", "Technical", "Speed", "Strength",
"Endurance"]]
y_new = new_data["Transfer Value (mil €)"]


# Splitting the data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
X_combined = pd.concat([X_train, X_new], ignore_index=True)
y_combined = pd.concat([y_train, y_new], ignore_index=True)

# Scaling data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

print(f"Original Dataset Size: {data.shape}")
print(f"Training Data Size: {X_train.shape}")
print(f"Test Data Size: {X_test.shape}")

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

# Building and training the model
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)

# Making predictions on the test set
y_pred = model.predict(X_test)

# Calculating performance metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("R-squared (R²):", r2)


#1 Comparing actual and estimated transfer values
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.7)
```

```
plt.xlabel("Real Transfer Value (€)")
plt.ylabel("Estimated Transfer Value (€)")
plt.title("Real vs. Estimated Transfer Value (€)")
plt.show()

sample_players = X_test[:5]
predicted_values = model.predict(sample_players)

print("Sample Player Predictions:")
for i, value in enumerate(predicted_values):
    print(f"Player {i+1}: Predicted Transfer Value = {value:.2f} million
€")


#2 Correlation of Features with Transfer Value
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv("extended_player_data.csv")

numeric_data = data.select_dtypes(include=['float64', 'int64'])

correlation = numeric_data.corr()["Transfer Value (mil €)"][:-1]

features = correlation.index
weights = [abs(w) for w in correlation.values]

plt.figure(figsize=(8, 6))
plt.pie(weights, labels=features, autopct='%1.1f%%', startangle=140)
plt.title("Weights of Features According to Transfer Value")
plt.show()


#3 K-Means Clustering
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import seaborn as sns

X_cluster = new_data[["Finishing", "Pass", "Technical", "Speed",
"Strength", "Endurance"]]

kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X_cluster)

new_data["Cluster"] = clusters

plt.figure(figsize=(10, 6))
sns.scatterplot(data=new_data, x="Finishing", y="Speed", hue="Cluster",
palette="viridis")
plt.title("Clustering Player (Finishing vs Speed)")
plt.xlabel("Finishing")
plt.ylabel("Speed")
plt.legend(title="Cluster")
plt.show()


#4 Transfer Value Estimation with Random Forest
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
```

```
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train.to_numpy())

y_new_pred = rf_model.predict(X_new.to_numpy())

mae = mean_absolute_error(y_new, y_new_pred)
print(f"New Data Set MAE: {mae}")

plt.figure(figsize=(8, 6))
plt.scatter(y_new, y_new_pred, alpha=0.7)
plt.xlabel("Real Transfer Value (€)")
plt.ylabel("Estimated Transfer Value (€)")
plt.title("Real vs Estimated Transfer Value (Random Forest) (€)")
plt.show()


#5 Correlation Matrix
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

new_data = pd.read_csv("extended_player_data.csv")

numeric_data = new_data.select_dtypes(include=['float64', 'int64'])

correlation_matrix = numeric_data.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation of Performance Metrics")
plt.show()
```

Main 8

```
#Code: Education and Test Data Preparation

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Dataset
data = pd.read_csv("extended_player_data.csv")

# Separate features and target variable
X = data[["Finishing", "Pass", "Technical", "Speed", "Strength",
"Endurance"]]  # Performance metrics
y = data["Transfer Value (mil €)"]  # Target variable (transfer value)

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

# Scaling of data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#Code: Implementing Algorithms and Performance Evaluation

from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error, r2_score

# Defining models
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(random_state=42),
    "Gradient Boosting": GradientBoostingRegressor(random_state=42),
    "Support Vector Regression": SVR()
}

results = []

# Testing each model
for model_name, model in models.items():
    # Train the model
    model.fit(X_train, y_train)

    # Making predictions
    y_pred = model.predict(X_test)

    # Calculating performance metrics
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    results.append({"Model": model_name, "MAE": mae, "R²": r2})

# Displaying results in a DataFrame
results_df = pd.DataFrame(results)
print(results_df)

# Visualizing feature importance using Random Forest

import matplotlib.pyplot as plt
```

```
feature_importances = models["Random Forest"].feature_importances_
features = X.columns

plt.figure(figsize=(8, 6))
plt.barh(features, feature_importances, color='skyblue')
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.title("Feature Importance in Random Forest")
plt.show()
```

Main 9

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

# Sample dataset
data = pd.read_csv("extended_player_data.csv")

# Separating feature and target variables
X = data[["Finishing", "Pass", "Technical", "Speed", "Strength",
"Endurance"]]
y = data["Transfer Value (mil €)"]

# Training the Random Forest model
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X, y)

# Extracting feature importances
feature_importances = rf_model.feature_importances_
features = X.columns

# Visualizing feature importances
plt.figure(figsize=(8, 6))
plt.barh(features, feature_importances, color='skyblue')
plt.xlabel("Features Importance")
plt.ylabel("Features")
plt.title("Feature Importance Analysis using Random Forest")
plt.show()


from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error

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

# Retraining the Random Forest model
rf_model.fit(X_train, y_train)

# Making predictions
y_pred = rf_model.predict(X_test)

# Mean Absolute Error calculation
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error (MAE): {mae}")

# Code snippet to generate the scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.7, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--',
lw=2)
plt.xlabel("Actual Transfer Value (€)")
plt.ylabel("Predicted Transfer Value (€)")
plt.title("Comparison of Actual vs Predicted Transfer Values")
plt.show()
```