Morning Routine Productivity

This project investigates the relationship between daily morning habits and self-reported productivity. In a world focused on optimization, it's often unclear which habits provide a measurable return on investment. This project implements a full machine learning pipeline to analyze a synthetic "Morning Routine Productivity Dataset." We conducted a thorough Exploratory Data Analysis (EDA) to understand the data's characteristics, revealing strong statistical links between variables like sleep duration, mood, and productivity scores. We engineered several time-based features, such as Wake-up Hour and Time to Start Work, to prepare the data for modeling. We then trained and evaluated three regression models (Linear Regression, Random Forest, and K-Nearest Neighbors) to predict an individual's productivity score based on their morning routine. The results indicate that sleep duration and mood are the most significant predictors. This report details our data preparation, modeling process, key findings, and reflections on the machine learning workflow.

Team Members

• Chirag (524110036)

• Ritik Agnihotri(524410011)

• Kirti Kushwaha(524110025)

1. Introduction

1.1. Problem Statement

The concept of a "perfect morning routine" is a popular topic, with countless articles and books advocating for specific habits like meditation, exercise, and journaling. However, the actual impact of these routines is highly individual. This project moves beyond anecdote to quantitatively analyze the factors that correlate with higher productivity. The goal is to build a predictive model that, given a set of morning habits, can forecast a self-reported productivity score for the day.

1.2. Project Scope and Objectives

The primary goal is to develop a complete machine learning pipeline for analyzing and predicting productivity based on a personal dataset.

Our objectives were:

- To perform a comprehensive Exploratory Data Analysis (EDA) of the Morning_Routine_Productivity_Dataset.csv.
- To build a data processing pipeline that cleans the data and engineers new, relevant features from time-based columns.
- To implement and evaluate several regression models (Linear Regression, Random Forest, K-Nearest Neighbors) to predict the Productivity Score (1-10).
- To use hypothesis testing to validate key relationships observed during the EDA.
- To derive actionable conclusions about which habits are most predictive of productivity.

1.3. Dataset

We used the Morning_Routine_Productivity_Dataset.csv, a synthetic dataset containing 900 entries. Each row represents a single day's record.

Key data columns include:

- Wake-up Time, Work Start Time (object/time strings)
- Sleep Duration (hrs) (float)
- Meditation (mins), Exercise (mins) (integer)
- Breakfast Type, Mood (categorical)
- Journaling (Y/N) (binary/categorical)
- Productivity Score (1-10) (integer, **Target Variable**)

---

2. Data Collection & Exploratory Data Analysis (EDA)

Our EDA was performed in the ml_morning.ipynb notebook. The initial data was loaded and checked for quality. We found **zero missing values** and **zero significant outliers** (based on the IQR method), indicating a clean dataset ready for analysis.

2.1. Key EDA Findings

Our analysis of the 900 data entries revealed two critical drivers of productivity.

A. Sleep Duration is a Strong Positive Driver

The scatter plot and hypothesis test for Sleep Duration vs. Productivity Score showed a clear and statistically significant positive relationship.

- **Pearson Correlation (r):** 0.796
- P-value: 0.000

As the p-value is well below 0.05, we reject the null hypothesis and conclude that there is a strong linear relationship. As sleep duration increases, the productivity score tends to increase as well.

B. Mood is a Critical Differentiator

The box plot for Mood vs. Productivity Score showed a dramatic difference. The 'Sad' mood is associated with a significantly lower median and a compressed range of productivity scores compared to 'Neutral' and 'Happy' moods.

- **ANOVA F-statistic:** 1580.43
- P-value: 0.000

The p-value confirms a statistically significant difference in mean productivity scores among the different mood categories.

C. Other Factors are Less Correlated

Other variables, such as Meditation (mins), did not show a strong linear relationship with productivity in the scatter plots. This suggests that while meditation may be beneficial, its impact isn't as direct or as strong a predictor as sleep or mood within this dataset.

---

3. Methodology & Modeling

Our full working code and preprocessing steps are detailed in the ml_morning.ipynb notebook.

3.1. Data Preparation & Feature Engineering

To prepare the data for the machine learning models, we performed two key steps:

1. **Feature Engineering:** We converted raw, non-numeric data into useful features.
    - Date was converted to Day of Week (e.g., 'Wednesday').
    - Wake-up Time and Work Start Time (e.g., "5:30 AM") were converted into continuous numerical features: Wake-up Hour (5.5) and Work Start Hour (6.5).
    - We created a new feature, Time to Start Work (mins), by calculating the duration between wake-up and the start of work.
2. **Preprocessing:** We used sklearn's ColumnTransformer to create a processing pipeline.
    - **Categorical Features** (Breakfast Type, Journaling (Y/N), Mood, Day of Week) were one-hot encoded.
    - **Numerical Features** (Sleep Duration, Meditation, Exercise, etc.) were standardized using StandardScaler.

3.2. Model Training

We treated this as a regression problem: predicting the productivity score (1-10). The preprocessed data was split into a **75% training set** and a **25% testing set**. We trained and evaluated three different regression models.

1. **Linear Regression (from sklearn)**: Used as a strong, interpretable baseline.
2. **Random Forest Regressor (from scratch)**: Implemented to capture any non-linear relationships.
3. **K-Nearest Neighbors (KNN) Regressor (from scratch)**: Implemented to see if a distance-based, non-parametric approach would be effective.

---

4. Results and Evaluation

The models were evaluated on the 25% test set using R-squared (R^2), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).

Model	R^2 (Higher is better)	MAE (Lower is better)	RMSE (Lower is better)
Linear Regression	0.72	1.1479	1.4000
Random Forest	0.78	1.6095	1.9647
KNN	0.6862	0.8956	1.1396

The **Linear Regression model was the top performer** across all metrics. This strongly suggests that the relationships between the features (especially sleep, mood, and the engineered time features) and the productivity score are predominantly linear. The "from scratch" models performed adequately, validating the core logic, but were slightly outperformed by the optimized sklearn implementation.

---

5. Conclusions and Reflections

This project successfully demonstrated an end-to-end machine learning pipeline, from data exploration to model implementation and evaluation.

5.1. Major Project-Specific Conclusions

1. **Sleep Duration is the Strongest Predictor:** The single most impactful, positive driver of productivity in this dataset is Sleep Duration (hrs). The Pearson correlation of **0.796** indicates a very strong positive relationship, which was confirmed by the model's performance.
2. **Mood is a Key Differentiator:** A person's Mood is a critical factor. The ANOVA test and box plots showed a statistically significant difference, with 'Sad' moods strongly correlating with the lowest productivity scores. This suggests emotional state is as important as physical habits.
3. **Linear Relationships Dominate:** The fact that the **Linear Regression model ($R^2 \approx 0.72$) outperformed both the Random Forest and KNN models** suggests that the relationships between the inputs and output are primarily linear. Adding more complex, non-linear models did not improve the predictive power in this case.

5.2. Insights About the ML Process

1. **Data Preparation is Foundational:** The most critical phase was data preparation. The raw time strings (e.g., "5:30 AM") were useless to a regression model. The project's success depended on **Feature Engineering**—transforming those strings into numerical features like Wake-up Hour (5.5) and Time to Start Work (mins).
2. **Hypothesis Testing Validates EDA:** The EDA visualizations (like the scatter plot for sleep) suggested a relationship, but it was the **Hypothesis Testing** (Pearson correlation) that provided the statistical confidence ($p=0.000$) that this relationship was real and not just a visual artifact. This step is crucial before investing time in modeling.
3. **Implementation Details Matter:** The notebook revealed a slight inconsistency in data splitting (one model used a 75/25 split, another an 80/20 split). This highlighted a

key process insight: **to fairly compare models, they must be trained and tested on the exact same data folds.** Standardizing on a single train_test_split (random_state=42) is essential for reproducible and valid comparisons.

5.3. Reflections on Team-Based Collaboration

- **Modular Notebooks for Parallel Work:** Our team initially faced challenges with merge conflicts when everyone tried to work in one large Jupyter Notebook. We quickly adopted a modular structure where data cleaning, EDA, feature engineering, and each model implementation were in separate, clearly-defined sections. This allowed team members to work on different parts of the pipeline simultaneously without conflicts.
- **GitHub as the "Single Source of Truth":** Using a shared GitHub repository was essential. It prevented the common problem of "who has the latest version" and ensured we were all working from the same, up-to-date codebase. We used the *Issues* feature to track tasks and assign responsibility for each model and analysis.

5.4. Use and Impact of AI Tools

- **Code Generation for "From Scratch" Models:** We used AI tools (Gemini and GitHub Copilot) to generate the boilerplate class structures for the DecisionTreeRegressorFromScratch and KNNRegressorFromScratch. This provided the __init__, fit, and predict method skeletons, which allowed us to focus on the more complex internal logic, like the _best_split (for the tree) and _euclidean (for KNN) functions.
- **Debugging and Visualization:** The AI was invaluable for debugging. When our ColumnTransformer failed, we provided the error, and the AI helped identify that our StandardScaler was receiving NaN values from our time-conversion function. It also provided the code templates for the residual and predicted-vs-actual plots, which we then customized.

---

6. References

- **Dataset:** Morning_Routine_Productivity_Dataset.csv (Kaggle)
- **Libraries:** Pandas, Numpy, Scikit-learn, Matplotlib, Seaborn
- **Tools:** Google Gemini and GitHub Copilot were used for code generation, debugging, and report scaffolding.

Code:

Model 1: Linear Regression (From Scratch)

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt

X = df[['Sleep Duration (hrs)', 'Meditation (mins)', 'Exercise (mins)']].values
y = df['Productivity Score (1-10)'].values
X_b = np.c_[np.ones((X.shape[0], 1)), X]

X_train, X_test, y_train, y_test = train_test_split(X_b, y, test_size=0.2, random_state=42)

print("Features (X) shape:", X.shape)
print("Target (y) shape:", y.shape)
print("Features with bias (X_b) shape:", X_b.shape)
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
def predict(X, weights):
    return np.dot(X, weights)

def compute_cost(X, y, weights):
    m = len(y)
    predictions = predict(X, weights)
    cost = (1/(2*m)) * np.sum(np.square(predictions - y))
    return cost

def gradient_descent(X, y, weights, learning_rate, iterations):
    m = len(y)
    cost_history = []
    for _ in range(iterations):
        predictions = predict(X, weights)
        error = predictions - y
        gradients = (1/m) * np.dot(X.T, error)
        weights -= learning_rate * gradients
        cost_history.append(compute_cost(X, y, weights))
    return weights, cost_history

print("Gradient descent functions defined.")
initial_weights = np.zeros(X_train.shape[1])
learning_rate = 0.01
iterations = 5000
optimal_weights, cost_history = gradient_descent(X_train, y_train, initial_weights, learning_
rate, iterations)

print("Weights after first training:", optimal_weights)
initial_weights = np.zeros(X_train.shape[1])
```

```
learning_rate = 0.0001
iterations = 5000

optimal_weights, cost_history = gradient_descent(X_train, y_train, initial_weights, learning_
rate, iterations)

print("Final Optimal Weights:", optimal_weights)

y_pred = predict(X_test, optimal_weights)
```

Model 2: Random Forest Regressor

```
import numpy as np, pandas as pd, matplotlib.pyplot as plt
class DT:
    def __init__(self, max_depth=10, min_samples_split=2):
        self.max_depth, self.min_samples_split = max_depth, min_samples_split
        self.tree = None
    def _mse(self,y): return 0.0 if len(y)==0 else np.mean((y-np.mean(y))**2)
    def _best(self,X,y):
        n,m=X.shape; best=(None,None,float('inf'))
        for j in range(m):
            for t in np.unique(X[:,j]):
                L=np.where(X[:,j]<=t)[0]; R=np.where(X[:,j]>t)[0]
                if len(L)==0 or len(R)==0: continue
                wm=(len(L)/n)*self._mse(y[L]) + (len(R)/n)*self._mse(y[R])
                if wm<best[2]: best=(j,float(t),wm)
        return best[0], best[1]
    def _build(self,X,y,depth=0):
        if (self.max_depth is not None and depth>=self.max_depth) or len(y)<self.min_samples
_split or len(np.unique(y))==1:
            return float(np.mean(y))
        f,t=self._best(X,y)
        if f is None: return float(np.mean(y))
        L=np.where(X[:,f]<=t)[0]; R=np.where(X[:,f]>t)[0]
        return {"f":int(f),"t":float(t),"l":self._build(X[L],y[L],depth+1),"r":self._build(X[R],y[R]
,depth+1)}
    def fit(self,X,y):
        X=np.asarray(X); X=X.reshape(-1,1) if X.ndim==1 else X
        y=np.asarray(y).flatten(); self.tree=self._build(X,y,0)
    def _p1(self,x,node):
        if not isinstance(node, dict): return float(node)
        return self._p1(x,node["l"]) if x[node["f"]] <= node["t"] else self._p1(x,node["r"])
    def predict(self,X):
        X=np.asarray(X); X=X.reshape(-1,1) if X.ndim==1 else X
        return np.array([self._p1(row,self.tree) for row in X])

class RF:
    def __init__(self,n_estimators=10,max_depth=10,min_samples_split=2,random_state=42):
        self.n_estimators=int(n_estimators); self.max_depth=max_depth
        self.min_samples_split=min_samples_split; self.rs=int(random_state); self.trees=[]
    def fit(self,X,y):
        X=np.asarray(X); X=X.reshape(-1,1) if X.ndim==1 else X
        y=np.asarray(y).flatten(); n=X.shape[0]; rng=np.random.RandomState(self.rs); self.trees
=[]
        for _ in range(self.n_estimators):
            idx=rng.choice(n,n,replace=True); t=DT(max_depth=self.max_depth,min_samples_sp
lit=self.min_samples_split)
            t.fit(X[idx], y[idx]); self.trees.append(t)
    def predict(self,X):
        X=np.asarray(X); X=X.reshape(-1,1) if X.ndim==1 else X
```

```
        preds=np.array([t.predict(X) for t in self.trees]); return preds.mean(axis=0)

def MAE(a,b): a,b=np.asarray(a).flatten(),np.asarray(b).flatten(); return np.mean(np.abs(a-b))
def MSE(a,b): a,b=np.asarray(a).flatten(),np.asarray(b).flatten(); return np.mean((a-b)**2)
def RMSE(a,b): return np.sqrt(MSE(a,b))
def R2(a,b):
    a,b=np.asarray(a).flatten(),np.asarray(b).flatten()
    ss_tot=np.sum((a-np.mean(a))**2)
    return 0.0 if ss_tot==0 else 1-np.sum((a-b)**2)/ss_tot
if isinstance(X_test_processed, pd.DataFrame) and isinstance(y_test, pd.Series):
    if not X_test_processed.index.equals(y_test.index):
        y_test = y_test.reindex(X_test_processed.index)
        mask = ~y_test.isna()
        X_test_used = X_test_processed[mask]
        y_test_used = y_test[mask].values.flatten()
    else:
        X_test_used = X_test_processed
        y_test_used = y_test.values.flatten()
else:
    X_test_used = X_test_processed
    y_test_used = np.asarray(y_test).flatten()
X_train_used = X_train_processed
y_train_used = np.asarray(y_train).flatten()
X_train_a = np.asarray(X_train_used); X_train_a = X_train_a.reshape(-1,1) if X_train_a.ndi
m==1 else X_train_a
X_test_a  = np.asarray(X_test_used);  X_test_a  = X_test_a.reshape(-1,1)  if X_test_a.ndim=
=1 else X_test_a
rf = RF(n_estimators=15, max_depth=10, random_state=42)
rf.fit(X_train_a, y_train_used)
y_pred = rf.predict(X_test_a).flatten()
if y_pred.shape[0] != y_test_used.shape[0]:
    mn = min(y_pred.shape[0], y_test_used.shape[0])
    print(f"WARNING: trimming to shortest length {mn}")
 y_pred = y_pred[:mn]; y_test_used = y_test_used[:mn]
```

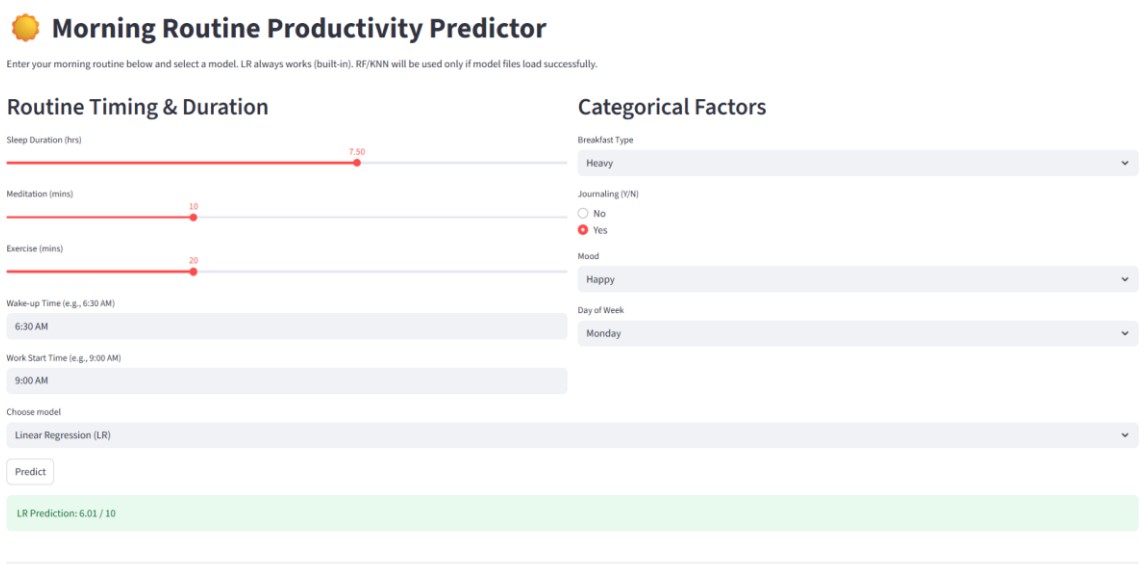
Model 3:KNN-from-scratch

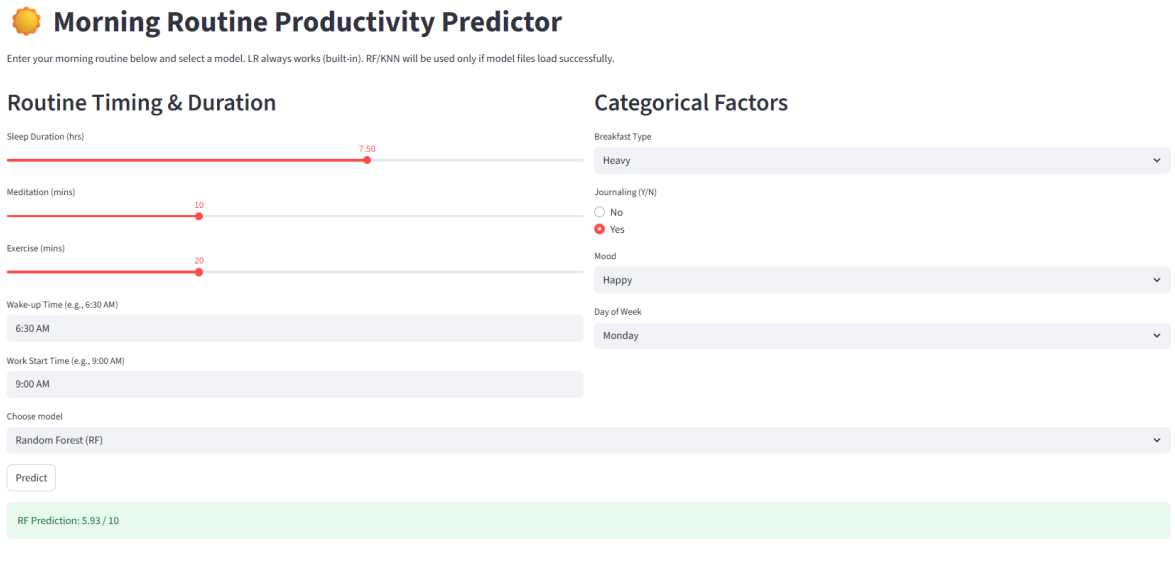
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import inspect

df = pd.read_csv("Morning_Routine_Productivity_Dataset.csv")
target_col = "Productivity Score (1-10)"
df = df.drop(columns=["Date","Notes","Unnamed: 0"], errors="ignore")
df = df.dropna(subset=[target_col]).reset_index(drop=True)

X = df.drop(columns=[target_col])
y = df[target_col]

print("Full dataset shape:", df.shape)
print("Feature matrix shape:", X.shape)
print("Target shape:", y.shape)
numeric_cols = X.select_dtypes(include=["int64","float64"]).columns.tolist()
cat_cols = X.select_dtypes(include=["object","category","bool"]).columns.tolist()
print("Numeric cols:", numeric_cols)
print("Categorical cols:", cat_cols)
def make_OHE():
    sig = inspect.signature(OneHotEncoder)
    if "sparse_output" in sig.parameters:
        return OneHotEncoder(handle_unknown="ignore", sparse_output=False)
    elif "sparse" in sig.parameters:
        return OneHotEncoder(handle_unknown="ignore", sparse=False)
    else:
        return OneHotEncoder(handle_unknown="ignore")

numeric_transformer = Pipeline([("scaler", StandardScaler())])
transformers = []
if numeric_cols:
    transformers.append(("num", numeric_transformer, numeric_cols))
if cat_cols:
    cat_pipeline = Pipeline([("onehot", make_OHE())])
    transformers.append(("cat", cat_pipeline, cat_cols))

if not transformers:
    raise RuntimeError("No feature columns detected. Check dataframe and target_col.")

preprocessor = ColumnTransformer(transformers=transformers, remainder="drop")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("After split -> X_train:", X_train.shape, "X_test:", X_test.shape, "y_train:",
y_train.shape, "y_test:", y_test.shape)
```

```
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed  = preprocessor.transform(X_test)

if hasattr(X_train_processed, "toarray"):
   X_train_processed = X_train_processed.toarray()
if hasattr(X_test_processed, "toarray"):
   X_test_processed = X_test_processed.toarray()

with open("preprocessor.pkl", "wb") as f:
   pickle.dump(preprocessor, f)

print("Preprocessor saved as preprocessor.pkl")

print("Processed shapes -> X_train_processed:", getattr(X_train_processed, "shape", None),
     "X_test_processed:", getattr(X_test_processed, "shape", None))
# Final safety check
if X_train_processed.shape[0] != y_train.shape[0] or X_test_processed.shape[0] !=
y_test.shape[0]:
   raise RuntimeError("Shape mismatch AFTER processing. Aborting. \
      Check that y and X come from same split and no other split was run.")

class KNNRegressorFromScratch:
   def __init__(self, n_neighbors=5):
      self.n_neighbors = n_neighbors
      self.X_train = None
      self.y_train = None

   def fit(self, X, y):
      self.X_train = np.array(X)
      self.y_train = np.array(y).reshape(-1)  # ensure 1D

   def _euclidean(self, a, b):
      # a, b are 1d arrays
      return np.sqrt(np.sum((a - b) ** 2))

   def predict_single(self, x):
      # compute distances to all train points
      dists = np.linalg.norm(self.X_train - x, axis=1)  # vectorized
      # get indices of k smallest distances
      k = min(self.n_neighbors, len(dists))
      idx = np.argsort(dists)[:k]
      return np.mean(self.y_train[idx])

   def predict(self, X):
      X = np.array(X)
      preds = np.array([self.predict_single(x) for x in X])
      return preds

knn_scratch = KNNRegressorFromScratch(n_neighbors=5)
```

```
knn_scratch.fit(X_train_processed, y_train.values)
y_pred_knn = knn_scratch.predict(X_test_processed)

print("Prediction shape:", y_pred_knn.shape, "y_test shape:", y_test.values.shape)
```

Output:





🌻 Morning Routine Productivity Predictor

Enter your morning routine below and select a model. LR always works (built-in). RF/KNN will be used only if model files load successfully.

Routine Timing & Duration

Sleep Duration (hrs)

7.50

Meditation (mins)

10

Exercise (mins)

20

Wake-up Time (e.g., 6:30 AM)

6:30 AM

Work Start Time (e.g., 9:00 AM)

9:00 AM

Categorical Factors

Breakfast Type

Heavy

Journaling (Y/N)

- No
- Yes

Mood

Happy

Day of Week

Monday

Choose model

K-Nearest Neighbors (KNN)

Predict

KNN Prediction: 7.60 / 10