📱 Mobile Price Prediction 📱

📉 by Decision Tree, Random Forest & SVM 📈

Members : MOKSHA BHANDARI , HET AMIN

1 - Libraries 📚

In [129…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.svm import SVC
from sklearn.datasets import make_classification
from sklearn import preprocessing
```

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
```

2 - Importing Dataset 🗄

In [130…
```
data = pd.read_csv('CellPhone_train.csv')
data
```

Out[130…

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobil
0	842	0	2.2	0	1	0	7	0.6	
1	1021	1	0.5	1	0	1	53	0.7	
2	563	1	0.5	1	2	1	41	0.9	
3	615	1	2.5	0	0	0	10	0.8	
4	1821	1	1.2	0	13	1	44	0.6	
...	...	...	...	...	...	...	...	...	
1995	794	1	0.5	1	0	1	2	0.8	
1996	1965	1	2.6	1	0	0	39	0.2	
1997	1911	0	0.9	1	1	1	36	0.7	
1998	1512	0	0.9	0	4	1	46	0.1	
1999	510	1	2.0	1	5	1	45	0.9	

2000 rows × 21 columns

In [131…
```
df=pd.DataFrame(data)
df
```

Out[131...

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobil
0	842	0	2.2	0	1	0	7	0.6	
1	1021	1	0.5	1	0	1	53	0.7	
2	563	1	0.5	1	2	1	41	0.9	
3	615	1	2.5	0	0	0	10	0.8	
4	1821	1	1.2	0	13	1	44	0.6	
...	...	...	...	...	...	...	...	...	
1995	794	1	0.5	1	0	1	2	0.8	
1996	1965	1	2.6	1	0	0	39	0.2	
1997	1911	0	0.9	1	1	1	36	0.7	
1998	1512	0	0.9	0	4	1	46	0.1	
1999	510	1	2.0	1	5	1	45	0.9	

2000 rows × 21 columns

3 - Dataset Description 📖

Dataset Description:

Index	Variable	Description
1	**battery_power**	*The maximum energy capacity of the mobile phone's battery, measured in milliamp hours (mAh).*
2	**blue**	*Indicates the presence of Bluetooth functionality (1: Yes, 0: No).*
3	**clock_speed**	*The processing speed of the mobile phone's microprocessor, measured in gigahertz (GHz).*
4	**dual_sim**	*Indicates whether the mobile phone supports dual SIM cards (1: Yes, 0: No).*
5	**fc**	*The resolution of the front camera, measured in megapixels (MP).*
6	**four_g**	*Indicates whether the mobile phone supports 4G network connectivity (1: Yes, 0: No)..*

Index	Variable	Description
7	**int_memory**	*The internal storage capacity of the mobile phone, measured in gigabytes (GB).*
8	**m_dep**	*The thickness of the mobile phone, measured in centimeters (cm).*
9	**mobile_wt**	*The weight of the mobile phone, measured in grams (g).*
10	**n_cores**	*The number of cores in the mobile phone's processor.*
11	**pc**	*The resolution of the primary (rear) camera, measured in megapixels (MP).*
12	**px_height**	*The height of the screen's resolution, measured in pixels.*
13	**px_width**	*The width of the screen's resolution, measured in pixels.*
14	**ram**	*The amount of Random Access Memory (RAM) available in the mobile phone, measured in megabytes (MB).*
15	**sc_h**	*The height of the mobile phone's screen, measured in centimeters (cm).*
16	**sc_w**	*The width of the mobile phone's screen, measured in centimeters (cm).*
17	**talk_time**	*The maximum duration the phone can be used for talking on a single battery charge, measured in hours.*
18	**three_g**	*Indicates whether the mobile phone supports 3G network connectivity (1: Yes, 0: No).*
19	**touch_screen**	*Indicates whether the mobile phone has a touch-sensitive screen (1: Yes, 0: No).*
20	**wifi**	*Indicates whether the mobile phone supports Wi-Fi connectivity (1: Yes, 0: No).*
21	**price_range**	*The target variable indicating the price range of the mobile phone, with values: 0 (low cost), 1 (medium cost), 2 (high cost), and 3 (very high cost).*

In [132…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   battery_power  2000 non-null   int64  
 1   blue           2000 non-null   int64  
 2   clock_speed    2000 non-null   float64
 3   dual_sim       2000 non-null   int64  
 4   fc             2000 non-null   int64  
 5   four_g         2000 non-null   int64  
 6   int_memory     2000 non-null   int64  
 7   m_dep          2000 non-null   float64
 8   mobile_wt      2000 non-null   int64  
 9   n_cores        2000 non-null   int64  
 10  pc             2000 non-null   int64  
 11  px_height      2000 non-null   int64  
 12  px_width       2000 non-null   int64  
 13  ram            2000 non-null   int64  
 14  sc_h           2000 non-null   int64  
 15  sc_w           2000 non-null   int64  
 16  talk_time      2000 non-null   int64  
 17  three_g        2000 non-null   int64  
 18  touch_screen   2000 non-null   int64  
 19  wifi           2000 non-null   int64  
 20  price_range    2000 non-null   int64  
dtypes: float64(2), int64(19)
memory usage: 328.3 KB
```

In [133…
```
df.isna().sum()
```

Out[133…
```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```

In [134…
```
df.duplicated()
```

Out[134…
```
0       False
1       False
2       False
3       False
4       False
        ...
1995    False
1996    False
1997    False
1998    False
1999    False
Length: 2000, dtype: bool
```

In [135…
```
df.dropna(inplace=True)
df.drop_duplicates(inplace=True)
```

In [136…
```
df.describe(include='all')
```

Out[136…

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	20
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	

8 rows × 21 columns

4 - Preprocessing 🎓

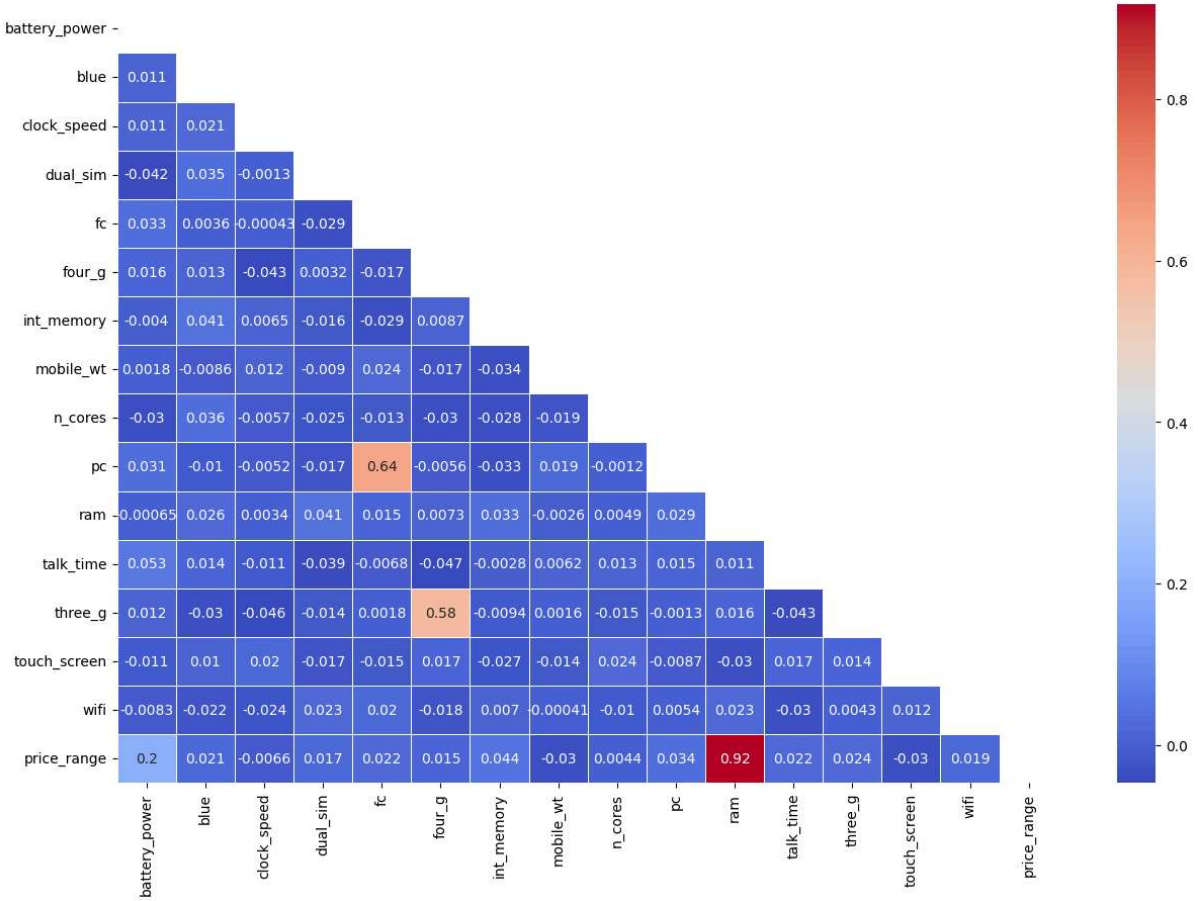
In [137…
```
df.drop(['px_height', 'px_width', 'sc_h', 'sc_w', 'm_dep'], axis=1, inplace=True)
```

In [138…
```
# price_range_counts = df['price_range'].value_counts()
# plt.figure(figsize=(4, 4))
# plt.pie(price_range_counts, labels=price_range_counts.index, autopct='%1.1f%%',
# plt.title('price_range Distribution')
# plt.axis('equal')
# plt.show()
```

```
# categorical_features = [ 'blue', 'wifi']
# for feature in categorical_features:
#     plt.figure(figsize=(5, 2.5))
#     sns.countplot(x=feature, hue='price_range', data=df, palette='viridis')
#     plt.title(f'Distribution of {feature} by price_range')
#     plt.show()
```

In [139…

```
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(numeric_only=True), annot=True,linewidth=.5,cmap="coolwarm",ma
```



In [140…

```
# categorical_features = ['blue','touch_screen', 'wifi' , 'four_g' , 'dual_sim']


# for feature in categorical_features:

#     feature_price_range = df.groupby([feature, 'price_range']).size().unstack().


#     for level in feature_price_range.index:
#         plt.figure(figsize=(4, 4))
#         plt.pie(feature_price_range.loc[level], labels=feature_price_range.colum
#         plt.title(f'price_range Distribution for {feature} = {level}')
#         plt.axis('equal')
#         plt.show()
```

In [141…

```
# sns.set(style="whitegrid")
```

```
# features1 = ['ram', 'battery_power', 'int_memory' , 'clock_speed']
# df[features1].hist(bins=15, figsize=(15, 10), layout=(2, 3) ,  color='green')
# plt.suptitle('Histogram of Features')
# plt.show()


# plt.figure(figsize=(15, 10))
# for i, feature in enumerate(features1, 1):
#     plt.subplot(2, 3, i)
#     sns.boxplot(y=df[feature])
#     plt.title(f'Box plot of {feature}')
# plt.tight_layout()
# plt.show()
```

In [142…

```
df
```

Out[142…

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	mobile_wt	r
0	842	0	2.2	0	1	0	7	188	
1	1021	1	0.5	1	0	1	53	136	
2	563	1	0.5	1	2	1	41	145	
3	615	1	2.5	0	0	0	10	131	
4	1821	1	1.2	0	13	1	44	141	
...	...	...	...	...	...	...	...	...	
1995	794	1	0.5	1	0	1	2	106	
1996	1965	1	2.6	1	0	0	39	187	
1997	1911	0	0.9	1	1	1	36	108	
1998	1512	0	0.9	0	4	1	46	145	
1999	510	1	2.0	1	5	1	45	168	

2000 rows × 16 columns

5 - Model Training without Hyperparameter Tunning

In [143…

```
X = df.drop('price_range', axis=1)
y = df['price_range']
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, ra



# Decision Tree
dt_model = DecisionTreeClassifier(random_state=0)
dt_model.fit(X_train, y_train)
dt_pred = dt_model.predict(X_test)
print("Decision Tree Accuracy:", accuracy_score(y_test, dt_pred))
print("Decision Tree Classification Report:\n", classification_report(y_test, dt_p

# Random Forest
rf_model = RandomForestClassifier(random_state=0)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, rf_pred))
print("Random Forest Classification Report:\n", classification_report(y_test, rf_p

# SVM
svm_model = SVC(random_state=0)
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)
print("SVM Accuracy:", accuracy_score(y_test, svm_pred))
print("SVM Classification Report:\n", classification_report(y_test, svm_pred))
```

```
Decision Tree Accuracy: 0.75
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.88      0.86        95
           1       0.68      0.64      0.66        92
           2       0.63      0.63      0.63        99
           3       0.83      0.83      0.83       114

    accuracy                           0.75       400
   macro avg       0.74      0.75      0.74       400
weighted avg       0.75      0.75      0.75       400

Random Forest Accuracy: 0.805
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.95      0.92        95
           1       0.71      0.73      0.72        92
           2       0.72      0.59      0.65        99
           3       0.85      0.94      0.89       114

    accuracy                           0.81       400
   macro avg       0.80      0.80      0.80       400
weighted avg       0.80      0.81      0.80       400

SVM Accuracy: 0.7825
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.89      0.90        95
           1       0.70      0.77      0.73        92
           2       0.66      0.65      0.65        99
           3       0.87      0.82      0.84       114

    accuracy                           0.78       400
   macro avg       0.78      0.78      0.78       400
weighted avg       0.79      0.78      0.78       400
```

6 - Model Training with Hyperparameter Tuning 📊

In [144…

```
X = df.drop('price_range', axis=1)
y = df['price_range']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, ra

# Decision Tree
dt_params = {'max_depth': [None, 10, 20, 30], 'min_samples_split': [2, 5, 10]}
dt_grid = GridSearchCV(DecisionTreeClassifier(random_state=0), dt_params, cv=5)
dt_grid.fit(X_train, y_train)
dt_best = dt_grid.best_estimator_
dt_pred = dt_best.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_pred)
print("Decision Tree Best Parameters:", dt_grid.best_params_)
print("Decision Tree Accuracy:", dt_accuracy)
print("Decision Tree Classification Report:\n", classification_report(y_test, dt_p

# Random Forest
rf_params = {'n_estimators': [100, 200, 300], 'max_depth': [None, 10, 20], 'min_sa
rf_grid = GridSearchCV(RandomForestClassifier(random_state=0), rf_params, cv=5)
rf_grid.fit(X_train, y_train)
rf_best = rf_grid.best_estimator_
rf_pred = rf_best.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_pred)
print("Random Forest Best Parameters:", rf_grid.best_params_)
print("Random Forest Accuracy:", rf_accuracy)
print("Random Forest Classification Report:\n", classification_report(y_test, rf_p

# SVM
svm_params = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf', 'poly'], 'gamma': ['s
svm_grid = GridSearchCV(SVC(random_state=0), svm_params, cv=5)
svm_grid.fit(X_train, y_train)
svm_best = svm_grid.best_estimator_
svm_pred = svm_best.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_pred)
print("SVM Best Parameters:", svm_grid.best_params_)
print("SVM Accuracy:", svm_accuracy)
print("SVM Classification Report:\n", classification_report(y_test, svm_pred))

# Compare Models
accuracies = {'Decision Tree': dt_accuracy, 'Random Forest': rf_accuracy, 'SVM': s
best_model = max(accuracies, key=accuracies.get)
print("Best Model:", best_model, "with accuracy:", accuracies[best_model])
```

```
Decision Tree Best Parameters: {'max_depth': None, 'min_samples_split': 10}
Decision Tree Accuracy: 0.76
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.88      0.87        95
           1       0.67      0.68      0.68        92
           2       0.65      0.61      0.63        99
           3       0.84      0.85      0.84       114

    accuracy                           0.76       400
   macro avg       0.75      0.76      0.75       400
weighted avg       0.76      0.76      0.76       400

Random Forest Best Parameters: {'max_depth': 20, 'min_samples_split': 10, 'n_estim
ators': 200}
Random Forest Accuracy: 0.785
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.92      0.91        95
           1       0.68      0.71      0.70        92
           2       0.69      0.58      0.63        99
           3       0.84      0.92      0.88       114

    accuracy                           0.79       400
   macro avg       0.78      0.78      0.78       400
weighted avg       0.78      0.79      0.78       400

SVM Best Parameters: {'C': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
SVM Accuracy: 0.83
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.95      0.93        95
           1       0.76      0.78      0.77        92
           2       0.78      0.63      0.70        99
           3       0.85      0.95      0.90       114

    accuracy                           0.83       400
   macro avg       0.83      0.83      0.82       400
weighted avg       0.83      0.83      0.83       400

Best Model: SVM with accuracy: 0.83
```

In [145…

```
data_test = pd.read_csv('CellPhone_test.csv')
data_test
```

Out[145...

	id	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep
0	1	1043	1	1.8	1	14	0	5	0.1
1	2	841	1	0.5	1	4	1	61	0.8
2	3	1807	1	2.8	0	1	0	27	0.9
3	4	1546	0	0.5	1	18	1	25	0.5
4	5	1434	0	1.4	0	11	1	49	0.5
...	...	...	...	...	...	...	...	...	...
995	996	1700	1	1.9	0	0	1	54	0.5
996	997	609	0	1.8	1	0	0	13	0.9
997	998	1185	0	1.4	0	1	1	8	0.5
998	999	1533	1	0.5	1	0	0	50	0.4
999	1000	1270	1	0.5	0	4	1	35	0.1

1000 rows × 21 columns

In [146...

```
dftest=pd.DataFrame(data_test)
dftest
```

Out[146...

	id	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep
0	1	1043	1	1.8	1	14	0	5	0.1
1	2	841	1	0.5	1	4	1	61	0.8
2	3	1807	1	2.8	0	1	0	27	0.9
3	4	1546	0	0.5	1	18	1	25	0.5
4	5	1434	0	1.4	0	11	1	49	0.5
...	...	...	...	...	...	...	...	...	...
995	996	1700	1	1.9	0	0	1	54	0.5
996	997	609	0	1.8	1	0	0	13	0.9
997	998	1185	0	1.4	0	1	1	8	0.5
998	999	1533	1	0.5	1	0	0	50	0.4
999	1000	1270	1	0.5	0	4	1	35	0.1

1000 rows × 21 columns

In [147…
```
dftest.dropna(inplace=True)
dftest.drop_duplicates(inplace=True)
dftest.drop(['px_height', 'px_width', 'sc_h', 'sc_w', 'm_dep' , 'id'], axis=1, inp
```

In [148…
```
X_test1 =dftest.copy()

X_test1_scaled = scaler.transform(X_test1)

dftest['price_test_with_decision_tree'] = dt_best.predict(X_test1_scaled)
dftest['price_test_with_random_forest'] = rf_best.predict(X_test1_scaled)
dftest['price_test_with_svm'] = svm_best.predict(X_test1_scaled)
```

In [149…
```
dftest
```

Out[149…

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	mobile_wt	n_
0	1043	1	1.8	1	14	0	5	193	
1	841	1	0.5	1	4	1	61	191	
2	1807	1	2.8	0	1	0	27	186	
3	1546	0	0.5	1	18	1	25	96	
4	1434	0	1.4	0	11	1	49	108	
…	…	…	…	…	…	…	…	…	
995	1700	1	1.9	0	0	1	54	170	
996	609	0	1.8	1	0	0	13	186	
997	1185	0	1.4	0	1	1	8	80	
998	1533	1	0.5	1	0	0	50	171	
999	1270	1	0.5	0	4	1	35	140	

1000 rows × 18 columns

In [150…
```
columns_to_extract = ['price_test_with_decision_tree', 'price_test_with_random_for
extracted_columns = dftest[columns_to_extract]
```

In [151…
```
extracted_columns
```

Out[151…

	price_test_with_decision_tree	price_test_with_random_forest	price_test_with_svm
0	3	3	3
1	3	3	3
2	2	2	2
3	3	3	3
4	1	1	1
...	...	...	...
995	2	2	2
996	1	1	1
997	1	0	1
998	2	2	2
999	2	2	2

1000 rows × 3 columns

In [152…
```
pip install flask
```

```
Requirement already satisfied: flask in c:\users\hetam\appdata\local\programs\pyth
on\python312\lib\site-packages (3.1.0)
Requirement already satisfied: Werkzeug>=3.1 in c:\users\hetam\appdata\local\progr
ams\python\python312\lib\site-packages (from flask) (3.1.3)
Requirement already satisfied: Jinja2>=3.1.2 in c:\users\hetam\appdata\local\progr
ams\python\python312\lib\site-packages (from flask) (3.1.4)
Requirement already satisfied: itsdangerous>=2.2 in c:\users\hetam\appdata\local\p
rograms\python\python312\lib\site-packages (from flask) (2.2.0)
Requirement already satisfied: click>=8.1.3 in c:\users\hetam\appdata\local\progra
ms\python\python312\lib\site-packages (from flask) (8.1.7)
Requirement already satisfied: blinker>=1.9 in c:\users\hetam\appdata\local\progra
ms\python\python312\lib\site-packages (from flask) (1.9.0)
Requirement already satisfied: colorama in c:\users\hetam\appdata\local\programs\p
ython\python312\lib\site-packages (from click>=8.1.3->flask) (0.4.6)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\hetam\appdata\local\pro
grams\python\python312\lib\site-packages (from Jinja2>=3.1.2->flask) (2.1.5)
Note: you may need to restart the kernel to use updated packages.
[notice] A new release of pip is available: 24.1.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [153…
```
pip install pickle-mixin
```

```
Requirement already satisfied: pickle-mixin in c:\users\hetam\appdata\local\progra
ms\python\python312\lib\site-packages (1.0.2)
Note: you may need to restart the kernel to use updated packages.
[notice] A new release of pip is available: 24.1.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [154…
```
import pickle
pickle.dump(dt_best,open('DT_Model.pkl','wb'))
pickle.dump(rf_best,open('RF_Model.pkl','wb'))
pickle.dump(svm_best,open('SVM_Model.pkl','wb'))
```

In [155…
```
import pickle

# Load models
dt_model = pickle.load(open('DT_Model.pkl', 'rb'))
rf_model = pickle.load(open('RF_Model.pkl', 'rb'))
svm_model = pickle.load(open('SVM_Model.pkl', 'rb'))
```

In [156…
```
X_test1.head()
```

Out[156…

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	mobile_wt	n_co
0	1043	1	1.8	1	14	0	5	193	
1	841	1	0.5	1	4	1	61	191	
2	1807	1	2.8	0	1	0	27	186	
3	1546	0	0.5	1	18	1	25	96	
4	1434	0	1.4	0	11	1	49	108	

In [157…
```
# Example: Using the loaded Decision Tree model for predictions

dt_prediction = dt_model.predict(X_test1_scaled)
print(f"DT Model Prediction: {dt_prediction}")
```

```
DT Model Prediction: [3 3 2 3 1 3 3 0 3 0 3 3 0 0 2 1 2 1 3 2 0 3 1 1 3 0 2 0 3 0
2 0 3 0 0 1 2
 1 3 1 1 2 0 0 0 1 1 2 1 3 2 0 3 0 3 1 2 1 1 3 3 3 0 2 1 1 1 2 1 1 1 2 2 3
 3 0 2 0 2 3 1 3 3 0 3 0 3 1 2 0 1 2 3 0 2 2 0 2 1 3 1 0 0 2 1 2 0 1 2 3 3
 2 1 3 3 3 3 1 3 0 0 3 1 1 1 0 3 3 2 2 0 2 2 1 3 0 2 0 3 2 1 3 1 2 3 3 3 2
 2 3 2 3 0 1 3 1 3 3 3 3 3 2 3 3 3 3 0 0 3 0 0 0 1 0 0 1 1 0 1 2 0 0 0 0 1
 2 1 0 0 0 0 0 0 3 1 1 2 2 2 3 1 2 2 3 2 1 2 0 0 0 1 2 1 1 3 2 0 2 0 3 2 3
 3 0 0 1 0 3 0 1 0 2 2 1 3 0 3 0 3 1 2 0 0 2 1 3 2 3 1 2 3 0 0 2 2 3 1 3 1
 0 3 2 1 2 3 3 3 1 0 1 2 2 1 1 3 2 0 3 0 1 2 0 0 3 2 3 3 3 1 3 2 2 3 2 2 1
 1 0 3 3 1 0 0 3 0 3 1 1 2 0 2 3 1 3 1 3 1 2 0 0 0 1 3 2 0 0 0 3 2 0 2 3 1
 3 3 2 3 1 3 3 1 2 2 3 3 0 3 1 3 1 3 2 3 3 0 1 1 3 1 3 2 3 0 0 0 0 2 0 0 2
 2 1 1 2 2 0 1 0 1 2 3 1 2 1 1 2 1 1 3 1 1 2 2 1 2 0 0 0 1 3 2 1 0 2 0 0 1
 1 0 1 0 2 2 3 2 3 0 3 1 3 0 2 1 1 1 0 2 2 3 2 1 3 2 3 2 3 2 1 2 2 2 1 0 0
 1 1 2 1 0 3 3 1 2 2 0 0 2 1 1 0 3 3 3 1 2 0 2 3 2 3 0 2 0 1 2 0 1 2 1 0 1
 1 2 3 3 2 2 3 1 1 2 2 2 3 2 1 3 2 2 2 1 0 2 2 0 0 0 3 1 2 2 2 3 1 3 0 2 2
 0 3 0 3 3 0 1 1 3 3 1 0 1 3 2 0 2 1 3 0 3 3 0 2 2 2 3 0 1 3 3 1 3 2 3 1 0
 1 0 3 1 0 3 2 3 2 0 2 3 2 2 3 2 1 2 1 2 3 3 0 0 1 1 1 2 2 0 0 2 2 3 2 0 3
 1 2 2 0 1 3 1 3 1 1 0 0 1 1 0 1 1 2 2 0 2 3 1 0 3 1 0 3 2 0 1 0 0 0 3 0 3
 1 3 2 1 3 2 0 2 1 3 1 3 2 1 2 0 1 1 2 0 1 1 1 1 2 2 3 0 3 2 1 1 3 2 0 1 2
 0 2 3 0 2 1 1 2 0 3 2 0 3 1 3 0 0 3 0 1 1 3 2 2 2 3 1 1 3 0 1 1 1 1 1 0 0
 1 0 0 2 0 1 2 0 1 0 0 3 0 3 3 3 0 0 2 2 2 1 0 1 2 0 1 1 0 1 3 3 2 3 1 2 3
 0 2 0 2 2 1 3 2 0 3 1 1 0 3 3 2 2 2 0 3 2 0 1 1 3 3 2 0 2 2 3 1 0 3 2 1 3
 1 2 2 1 1 3 1 1 2 1 0 0 2 2 0 2 0 0 1 1 2 3 3 3 0 1 2 2 1 0 0 3 1 0 2 0 2
 2 2 2 2 0 2 1 3 0 0 3 1 3 0 0 2 2 3 0 2 2 1 0 0 2 3 0 3 0 0 1 2 2 1 2 1 3
 2 1 2 3 3 0 1 1 1 1 2 2 0 1 2 1 1 3 1 2 2 1 1 1 3 3 3 0 3 3 0 2 3 2 2 3 3
 2 1 1 2 0 2 1 1 2 2 2 2 2 0 1 1 3 1 0 1 1 3 1 0 0 2 2 2 3 0 2 2 2 1 3 0 0
 2 1 2 1 1 3 3 0 3 0 3 2 0 3 2 2 2 1 0 2 3 1 0 2 1 3 2 3 0 2 2 0 2 3 2 3 0
 2 0 0 2 2 2 3 0 2 3 2 1 3 0 1 3 0 2 0 0 3 2 2 0 0 0 0 3 3 3 3 0 0 2 1 1 2
 2]
```

In [ ]: