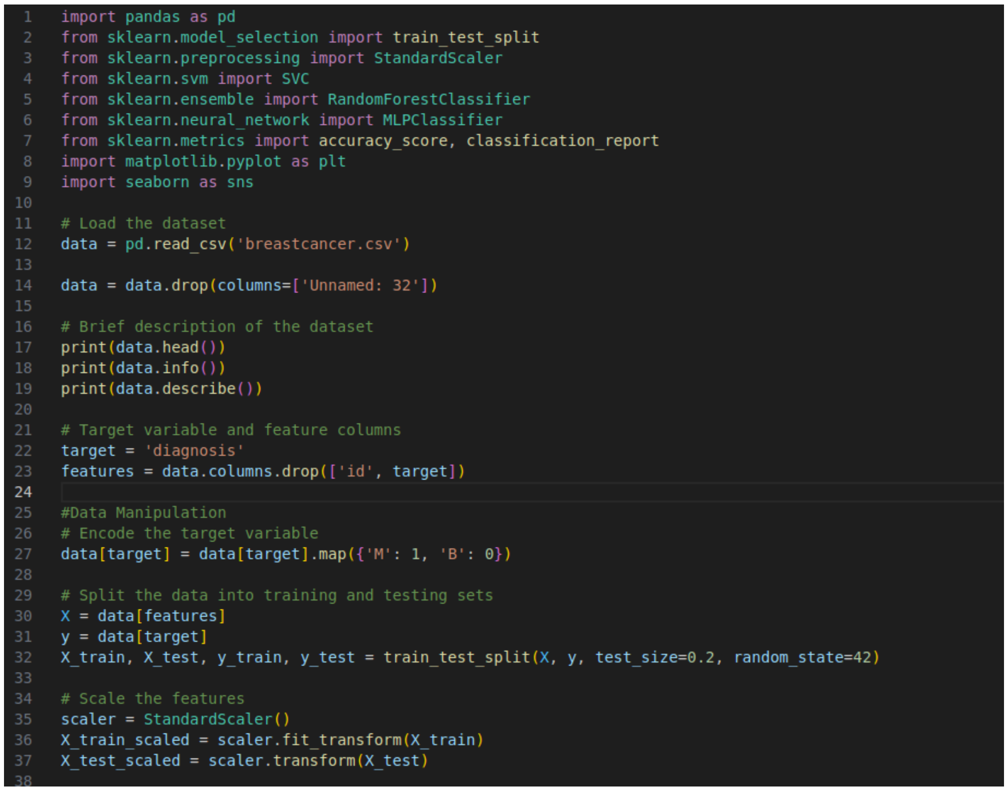
This document provides an overview of the Python script developed to analyze and compare the performance of different machine learning models for predicting cancer types based on a given dataset. The main sections of the script include data import, manipulation, model training, and result storage.

Data Description
The dataset used in this analysis is the Breast Cancer Wisconsin (Diagnostic) dataset. It contains 569 instances of breast cancer cases, each described by 32 features. The target variable is diagnosis, which indicates whether the cancer is malignant (M) or benign (B).

Columns:
id: Identifier for each instance
diagnosis: Target variable (M for malignant, B for benign)
30 numerical features describing the characteristics of the cell nuclei present in the image

Data Import and Manipulation
The dataset is imported using pandas, and the necessary preprocessing steps are performed, including encoding the target variable and scaling the features.



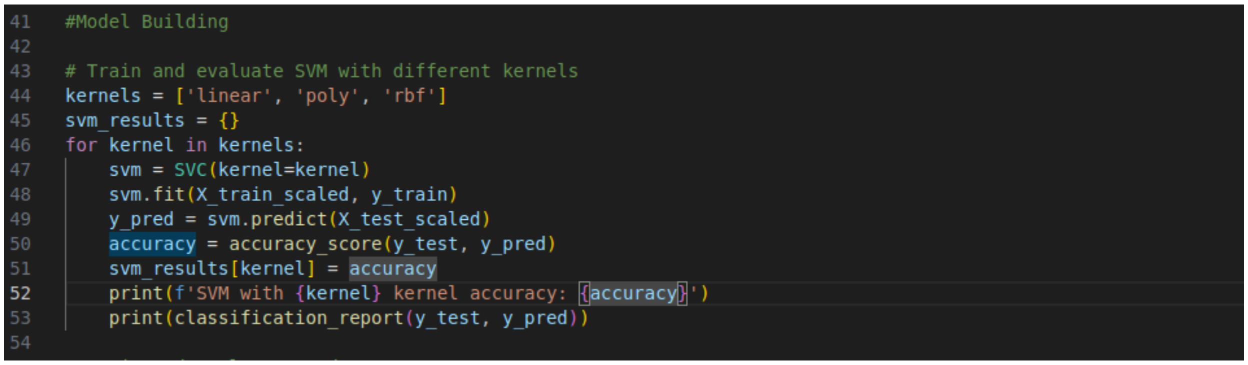
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
data = pd.read_csv('breastcancer.csv')

data = data.drop(columns=['Unnamed: 32'])

# Brief description of the dataset
print(data.head())
print(data.info())
print(data.describe())

# Target variable and feature columns
target = 'diagnosis'
features = data.columns.drop(['id', target])

#Data Manipulation
# Encode the target variable
data[target] = data[target].map({'M': 1, 'B': 0})

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

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Model Training

Three models were trained and evaluated: ***Support Vector Machines (SVM), Random Forest,*** and ***Neural Network***.

SVM Analysis

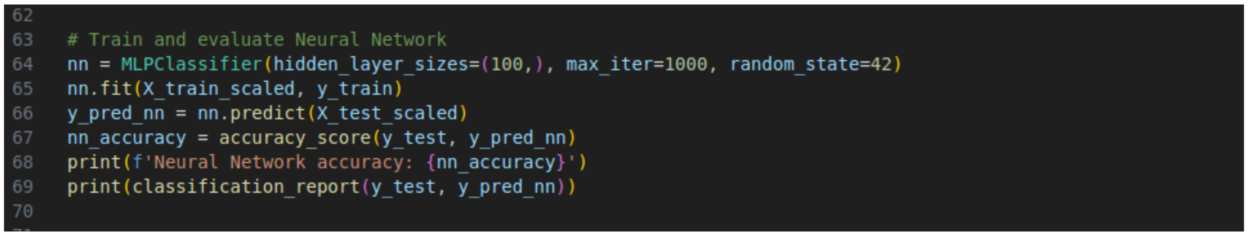
Support Vector Machines (SVM) are effective in high-dimensional spaces and versatile with different kernels. We used three different kernels: *linear*, *polynomial*, and *radial basis function (RBF)*.



```python
#Model Building

# Train and evaluate SVM with different kernels
kernels = ['linear', 'poly', 'rbf']
svm_results = {}
for kernel in kernels:
    svm = SVC(kernel=kernel)
    svm.fit(X_train_scaled, y_train)
    y_pred = svm.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    svm_results[kernel] = accuracy
    print(f'SVM with {kernel} kernel accuracy: {accuracy}')
    print(classification_report(y_test, y_pred))
```

Neural Network Regression Analysis

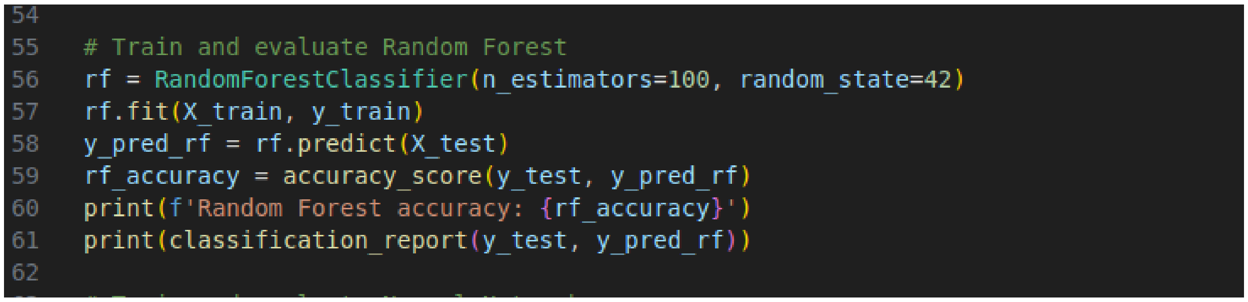
Neural networks are capable of *capturing complex patterns* and have a flexible architecture. We used a *Multi-Layer Perceptron (MLP)* classifier for this analysis.



```python
# Train and evaluate Neural Network
nn = MLPClassifier(hidden_layer_sizes=(100,), max_iter=1000, random_state=42)
nn.fit(X_train_scaled, y_train)
y_pred_nn = nn.predict(X_test_scaled)
nn_accuracy = accuracy_score(y_test, y_pred_nn)
print(f'Neural Network accuracy: {nn_accuracy}')
print(classification_report(y_test, y_pred_nn))
```

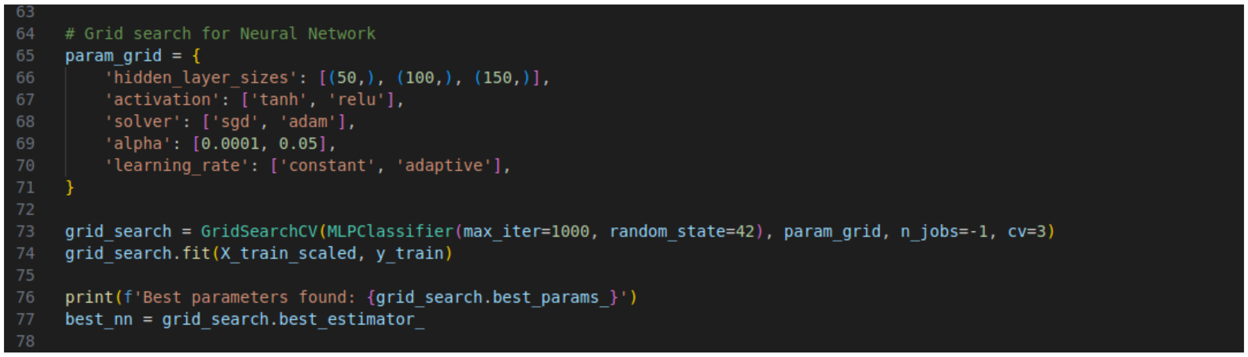
Random Forest Analysis

Random Forest is robust to *overfitting* and handles *large datasets* well. It was trained and evaluated as follows:



```python
# Train and evaluate Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
rf_accuracy = accuracy_score(y_test, y_pred_rf)
print(f'Random Forest accuracy: {rf_accuracy}')
print(classification_report(y_test, y_pred_rf))
```

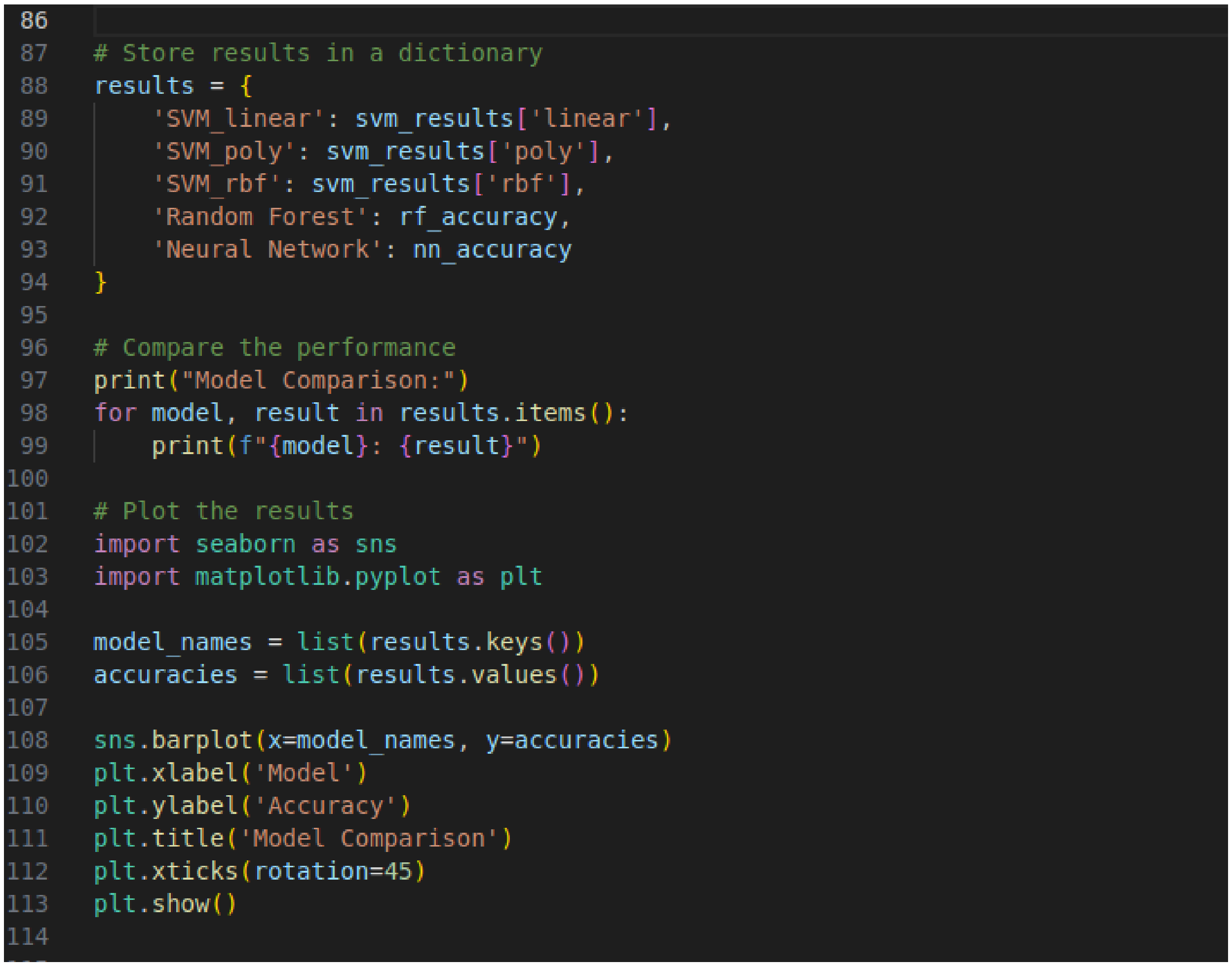
Grid Search for parameters



```python
# Grid search for Neural Network
param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (150,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant', 'adaptive'],
}

grid_search = GridSearchCV(MLPClassifier(max_iter=1000, random_state=42), param_grid, n_jobs=-1, cv=3)
grid_search.fit(X_train_scaled, y_train)

print(f'Best parameters found: {grid_search.best_params_}')
best_nn = grid_search.best_estimator_
```

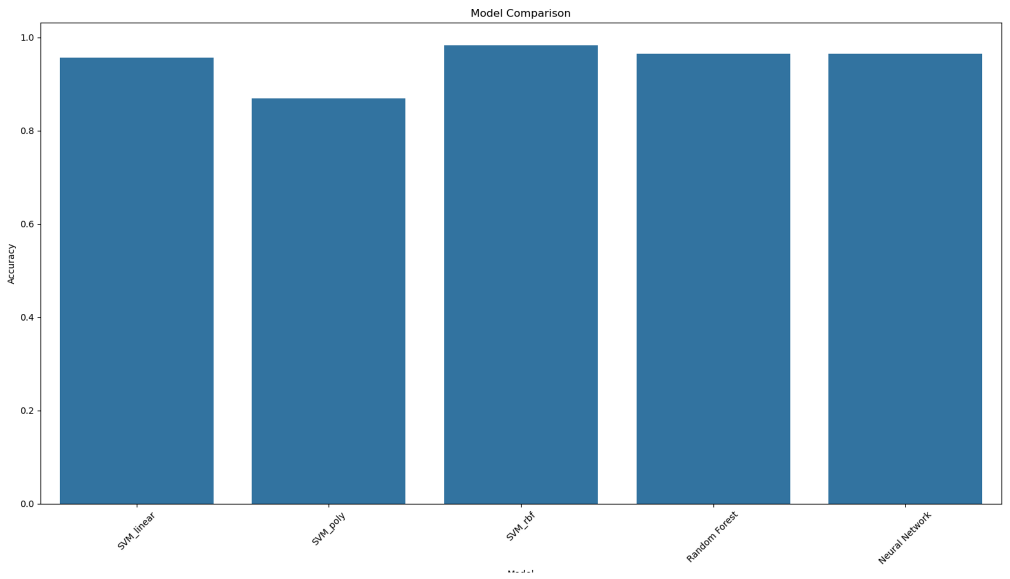
Comparison

The performance of the three models was compared based on their accuracy



```python

# Store results in a dictionary
results = {
    'SVM_linear': svm_results['linear'],
    'SVM_poly': svm_results['poly'],
    'SVM_rbf': svm_results['rbf'],
    'Random Forest': rf_accuracy,
    'Neural Network': nn_accuracy
}

# Compare the performance
print("Model Comparison:")
for model, result in results.items():
    print(f"{model}: {result}")

# Plot the results
import seaborn as sns
import matplotlib.pyplot as plt

model_names = list(results.keys())
accuracies = list(results.values())

sns.barplot(x=model_names, y=accuracies)
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Model Comparison')
plt.xticks(rotation=45)
plt.show()

```

Accuracy plot:



Accuracy of each models:

SVM with Linear kernel function:	0.956140350877193
SVM with Polynomial kernel function	0.868421052631579
SVM with Radial Basis function	0.9824561403508771
Random Forest	0.9649122807017544
Neural Network	0.9649122807017544

Discussion

Support Vector Machine (SVM)

- **Strengths**: SVMs are effective in high-dimensional spaces, making them suitable for complex classification problems. They are versatile, with options to apply different kernel functions to separate data that is not linearly separable.
- **Weaknesses**: SVMs can be computationally expensive, particularly with large datasets, and are sensitive to parameter settings, which can impact performance if not optimized carefully.

Random Forest

- **Strengths**: Random Forests are robust to overfitting, especially with complex datasets, and they handle large datasets well due to their ensemble approach.
- **Weaknesses**: Random Forest models can be less interpretable than simpler models and require careful parameter tuning, especially when dealing with high-dimensional data.

Neural Network

- **Strengths**: Neural Networks are highly flexible and can capture complex, non-linear patterns in data, which makes them powerful for various tasks in machine learning.
- **Weaknesses**: They often require large datasets to generalize well and are computationally intensive, requiring significant resources for both training and fine-tuning.

Real-World Applications

Accurate cancer type prediction has significant implications in the medical field. Early and accurate diagnosis can lead to better treatment plans and improved patient outcomes. Machine learning models, like the ones analyzed in this report, can assist doctors in making more informed decisions, potentially saving lives. These models can be integrated into diagnostic tools to provide real-time analysis and support to medical professionals.

References

kaggle.com
Paper on Benign and Malignant tumor cells
"Deep Learning" by Ian Goodfellow, Yoshua Bengio, and Aaron Courville
"Random Forests" by Leo Breiman (2001, *Machine Learning*)

Conclusion

The model with the highest accuracy is the **SVM with Radial Basis Function (RBF) kernel**, achieving an accuracy of 0.9825. This suggests that thoughtful model selection, combined with careful parameter tuning, is essential for achieving optimal performance in machine learning tasks.