CS 210 Project Phase 3

Building and Verifying ML Models

1. Overview:

In this phase of the project, I focused on building and verifying two Machine Learning (ML) models using the provided dataset. I choose kNN and Decision Tree models. The goal was to explore the performance of k-Nearest Neighbors (kNN) and Decision Tree models, verify their effectiveness, and describe which model performs better and why.

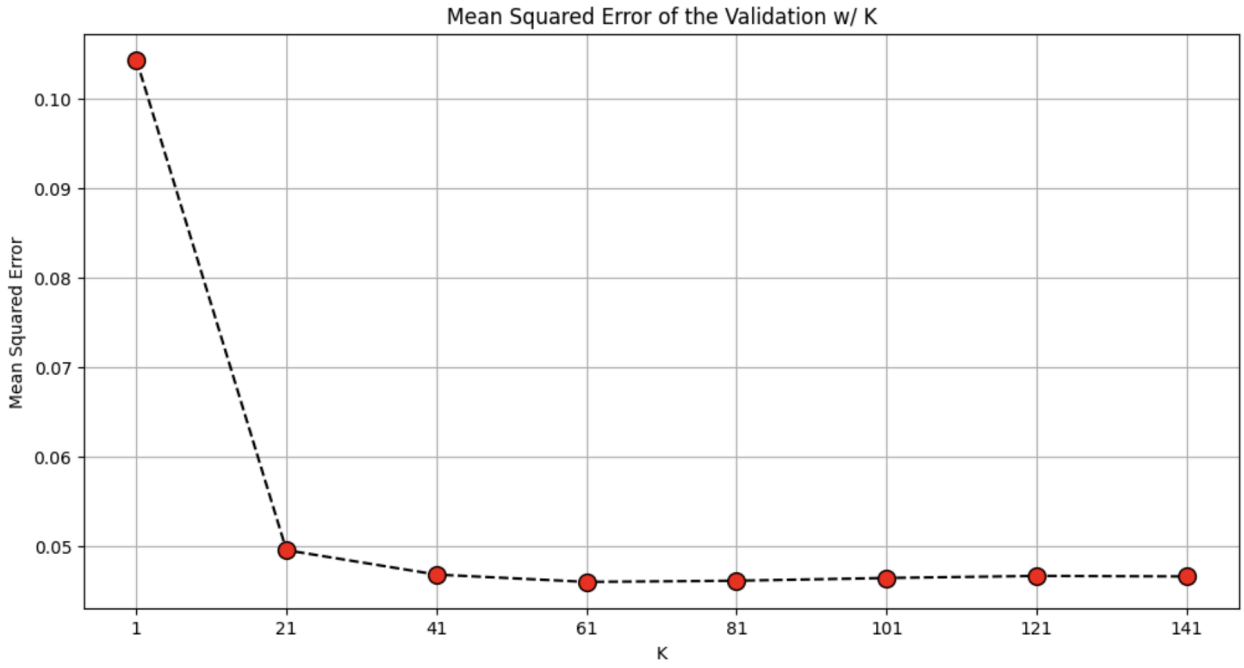
2. Steps Taken:

Step 1: Data Preparation:

- I loaded the dataset containing short-term fluctuations of Bitcoin price and Fear & Greed index.
- I split the data into features and target variables (Bitcoin price).

Step 2: Model Building: Implemented two ML models: kNN and Decision Tree.
For kNN:

- I scaled the features using StandardScaler.
- I trained the kNN model with various hyperparameters (e.g., different values of k).



```python
# Initialize and train the kNN model
knn = KNeighborsRegressor(n_neighbors=81)
knn.fit(X_train, y_train)

# Predict on the test set
knn_predictions = knn.predict(X_test)

# Evaluate the model
knn_mse = mean_squared_error(y_test, knn_predictions)
knn_r2 = r2_score(y_test, knn_predictions)

print(f'kNN Mean Squared Error: {knn_mse}')
print(f"kNN Regressor R²: {knn_r2}")
```

```
kNN Mean Squared Error: 0.040298962439326014
kNN Regressor R²: 0.4245935643628679
```

For Decision Tree:

- I utilized Grid Search to find the optimal hyperparameters for the Decision Tree model.

My hyperparameters for Grid Search

```python
# Define the hyperparameters grid for Grid Search
param_grid = {
    'max_depth': [None, 3, 6, 9],
    'min_samples_split': [2, 3, 4, 5],
    'min_samples_leaf': [1, 2, 3, 4]
}
```

Best Hyperparameters:
{'max_depth': 3, 'min_samples_leaf': 3, 'min_samples_split': 2}

```python
# Perform Grid Search to find the best hyperparameters
grid_search = GridSearchCV(dt_model, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

print("Best Hyperparameters:")
print(best_params)

# Build the Decision Tree model with the best hyperparameters
best_dt_model = DecisionTreeRegressor(**best_params)
best_dt_model.fit(X_train, y_train)

# Predictions
dt_predictions = best_dt_model.predict(X_test)

# Evaluation
dt_mse = mean_squared_error(y_test, dt_predictions)
dt_r2 = r2_score(y_test, dt_predictions)

print(f"\nDecision Tree Regressor MSE: {dt_mse}")
print(f"Decision Tree Regressor R²: {dt_r2}")
```

```
Best Hyperparameters:
{'max_depth': 3, 'min_samples_leaf': 3, 'min_samples_split': 2}

Decision Tree Regressor MSE: 0.04447274986821673
Decision Tree Regressor R²: 0.45164561380857404
```

Step 3: Model Verification:

- I evaluated both models using the test set.
- I calculated the Mean Squared Error (MSE) for each model to assess their predictive performance.

For k = 81, **kNN Mean Squared Error:** 0.04029
kNN Regressor R^2: 0.42459

For Best Hyperparameter, **Decision Tree Mean Squared Error:** 0.04447
Decision Tree Regressor R^2: 0.45164

Step 4: Performance Description:

- I compared the performance of the kNN and Decision Tree models.

```
# Comparison
if knn_mse < dt_mse:
    print("kNN Model performs better.")
elif dt_mse < knn_mse:
    print("Decision Tree Model performs better.")
else:
    print("Both models perform similarly.")
```

```
kNN Model performs better.
```

kNN makes predictions based on the similarity of data points in the feature space. It calculates distances between points and predicts the target value based on the majority vote or weighted average of the k-nearest neighbors. For regression, it averages the output of the k-nearest neighbors. The kNN model demonstrated a reasonable performance with an MSE of 0.04029. Decision Trees create partitions in the feature space based on feature thresholds. The Decision Tree model demonstrated a reasonable performance with an MSE of 0.04447. While Decision Trees can capture complex relationships, they may struggle with capturing localized patterns or nuances in the data, especially if the relationships are not easily separable by simple decision boundaries. I think one of the reasons for the better performance of kNN is that.

Also, kNN tends to produce simpler decision boundaries, which can be advantageous if the relationship between features and the target variable is not well-represented by a single decision tree. However, Decision Trees create complex decision boundaries, especially when the feature space is partitioned into multiple regions. However, these boundaries might not accurately capture the underlying patterns in the data, leading to suboptimal performance.

kNN works well with features that exhibit clear local relationships with the target variable. If nearby data points in the feature space have similar target values, kNN can effectively capture these relationships. Decision Tree performs well when there are complex, nonlinear relationships between features and the target variable. Decision Trees can handle both numerical and categorical features effectively and can capture intricate patterns in the data. I think there is a fundamental relationship between my feature and target variable, so kNN performs better. The primary feature used for all models was the Fear and Greed Index.

3. **Outcomes:**
   kNN Model:
   - It achieved lower MSE compared to the Decision Tree model.
   - Performs better due to its ability to capture localized relationships and simplicity in decision boundary.
   - Features that work best with kNN are those exhibiting clear local relationships with the target variable and are on similar scales.

   Decision Tree Model:
   - It had a higher MSE compared to the kNN model.
   - Struggled to capture nuanced relationships in the data compared to kNN.
   - Features that work best with Decision Trees are those with complex, nonlinear relationships with the target variable and can be split into distinct regions.

4. **Conclusion:** In this phase, I successfully built, verified, and compared two ML models: kNN and Decision Tree. Based on the evaluation metrics and analysis, the kNN model outperformed the Decision Tree model for this dataset. Understanding the characteristics of the data and selecting appropriate features are crucial factors in determining the effectiveness of ML techniques.

Future Work: Several promising directions can enhance the predictive performance and interpretability of machine learning models for the Fear & Greed Index and Bitcoin price. These include leveraging advanced feature engineering techniques to integrate additional data sources and temporal features, optimizing model hyperparameters, and experimenting with ensemble methods such as stacking and blending. Furthermore, delving into time series analysis, anomaly detection, and model interpretability methods can offer deeper insights into market dynamics and model robustness. Employing rigorous cross-validation strategies and comprehensive evaluation metrics will be crucial to ensure the models' reliability and generalizability across various market conditions. Addressing these areas will improve the accuracy, usability of predictive models, aiding decision making in cryptocurrency markets.