ML Experiment 9

Literature Survey

1. **Evolutionary f1 race strategy 2023**

   The research proposes a tailor-made genetic algorithm that employs data from free practice sessions to simulate and determine the optimal race strategy for a given circuit. The algorithm considers multiple factors that influence race performance, such as prevailing weather conditions, choice of tire compounds, timing and number of pit stops, fuel load, and tire degradation over the course of the race. By evaluating these variables through an evolutionary computational approach, the algorithm aims to identify the most advantageous strategy specific to the characteristics of the circuit under consideration.

2. **A Data-Driven Analysis of Formula 1 Car Races Outcome**

   The study adopts a data-driven methodology to determine the key factors that influence the overall points tally of Formula 1 drivers across an entire racing season. It employs correlation analysis techniques to establish relationships between various parameters and a driver's points haul. Furthermore, principal component analysis (PCA) is utilized to identify closely correlated factors and reduce the dimensionality of the data. By pinpointing the most significant contributors through this analytical approach, the research aims to provide insights into the elements that drive success over the course of a championship campaign.

3. **Online Planning for F1 Race Strategy Identification 2021**

   The research proposes an open-loop strategy that amalgamates Monte Carlo sampling techniques with Temporal Difference (TD) learning updates to optimally determine pit stop timings and tire compound selections during a race. This approach leverages probabilistic simulations through Monte Carlo methods while continuously improving decisions via TD updates based on the simulated outcomes. The efficacy of this methodology is evaluated using a sophisticated simulator that accurately models race dynamics by incorporating real-world data. By synergizing sampling and reinforcement learning concepts, the proposed solution aims to identify pit strategies that can potentially enhance overall race performance.

4. **Optimizing Pit Stops Strategies with Competition in a Zero-Sum Feedback Stackelberg Game 2023**

   The research models the pit stop strategy optimization problem as a two-player zero-sum feedback Stackelberg game, where two Formula 1 drivers competitors aim to

maximize their chances of winning. Dynamic Programming techniques are employed to solve this game theoretic formulation. The authors demonstrate that when one driver adopts a strategic, forward-looking approach while the opponent remains myopic and reactive, the strategic driver can boost their odds of victory by over 15%. This highlights the significant advantage that can be gained by carefully planning pit stop strategies while anticipating and adapting to the moves of rival competitors on the track.

5. **Poisson analysis**

   The research endeavored to develop predictive models for pit stop occurrences in Formula 1 races by utilizing lap data and employing Poisson regression models. However, the study encountered challenges such as over-dispersion in the data and issues with model convergence, which hindered accurate predictions. Consequently, the authors recommend a more focused approach, concentrating on a single race track while incorporating relevant weather data into the modeling process. By narrowing the scope to a specific circuit and accounting for meteorological factors, they anticipate circumventing the obstacles faced and achieving improved predictive performance for pit stop strategies.

6. **Gone in 2s: a deep dive into perfection analysing the collaborative maintenance pitstop of Formula 1 2021**

   The study conducts an in-depth examination of the rapid pit stop procedures in Formula 1 racing with the aim of developing a comprehensive set of guidelines for achieving efficient and effective maintenance practices. By employing fault tree analysis techniques, the research identifies potential points of failure or bottlenecks that can impede the smooth execution of pit stop operations. Drawing from these insights, the authors establish a framework outlining best practices that teams can adopt to streamline their maintenance procedures during pit stops, thereby minimizing downtimes and ensuring successful completion of all requisite tasks in a time-critical environment.

7. **Virtual strategy engineer: Using artificial neural networks for making race strategy decisions in circuit motorsport 2020**

   The research introduces the concept of a virtual strategy engineer (VSE), an artificial intelligence (AI) system tailored for Formula 1 racing. By training the VSE on comprehensive historical race data, encompassing factors such as track conditions, weather patterns, and competitor strategies, the system gains the capability to provide strategic recommendations pertaining to optimal timings for pit stops and ideal tire compound choices. The authors posit that the deployment of such an AI-driven strategic advisor could potentially revolutionize race strategy formulation, enabling teams to make more informed decisions that ultimately translate into improved overall performance and a competitive edge on the circuit.

8. **Rank position forecasting in car racing 2021**

   The study focuses on developing predictive models for forecasting the rank positions of cars during races. While acknowledging pit stops as a crucial factor influencing rank changes, the authors note the challenges in accurately predicting these events due to their irregular nature. To address this, they propose RankNet, an innovative deep learning architecture that decouples the analysis of rank dynamics from pit stop occurrences, treating them as separate predictive tasks. By incorporating uncertainty quantification techniques, RankNet demonstrates a significant improvement in anticipating future rank positions compared to traditional approaches. This advance paves the way for more reliable race strategy planning and decision-making by teams.

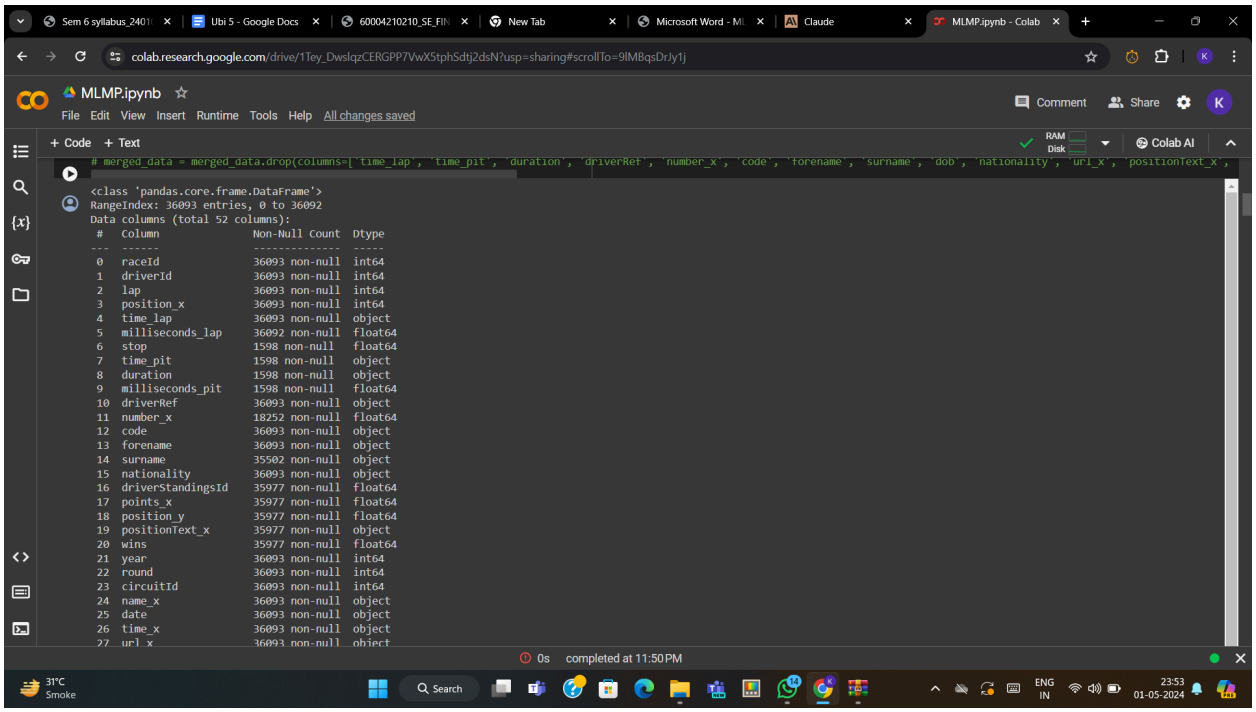
9. **Application of Monte Carlo methods to consider probabilistic effects in a race simulation for circuit motorsport 2020**

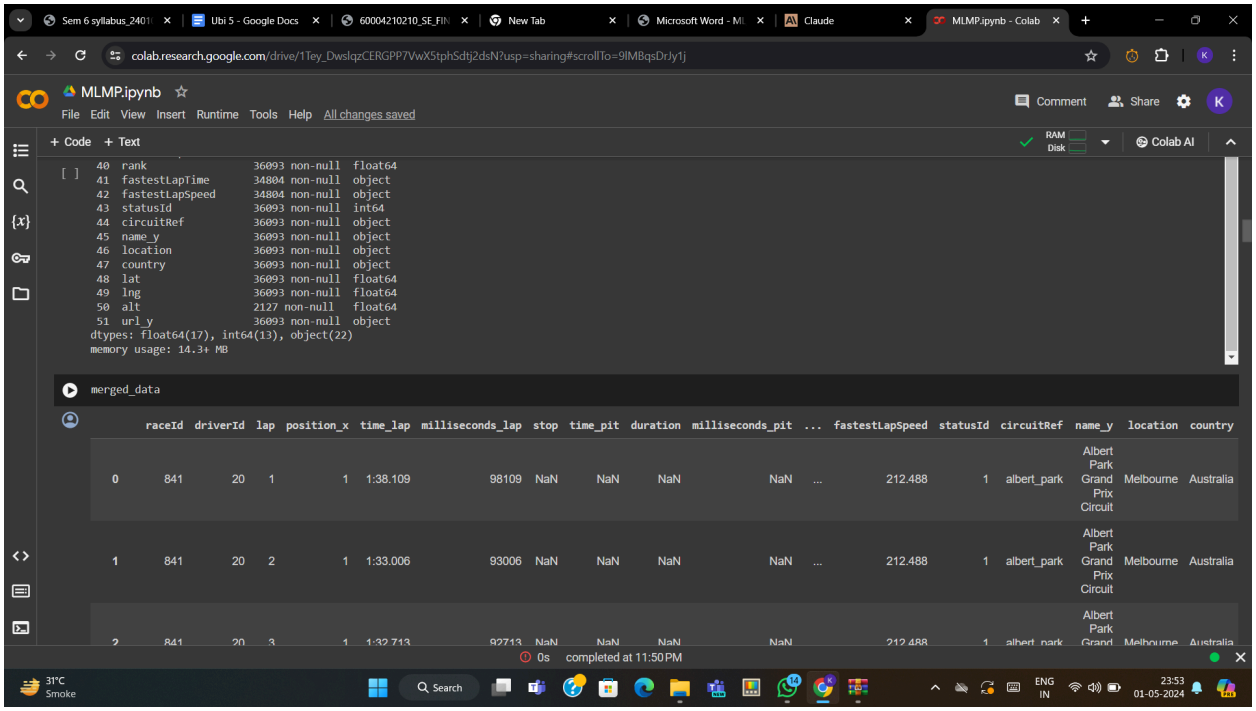
   The research enhances the realism and accuracy of race simulations in motorsport by accounting for probabilistic factors that can significantly impact race outcomes. It models stochastic events such as on-track accidents, safety car deployments, and variations in driver performance using tailored probability distributions. A novel "ghost car" concept is introduced to realistically simulate the effects of a safety car on the racetrack. By leveraging the Monte Carlo method to incorporate these probabilistic elements, the proposed approach enables teams to comprehensively evaluate the effectiveness of different strategic decisions under uncertain race conditions, facilitating more robust strategy formulation and preparedness for contingencies.

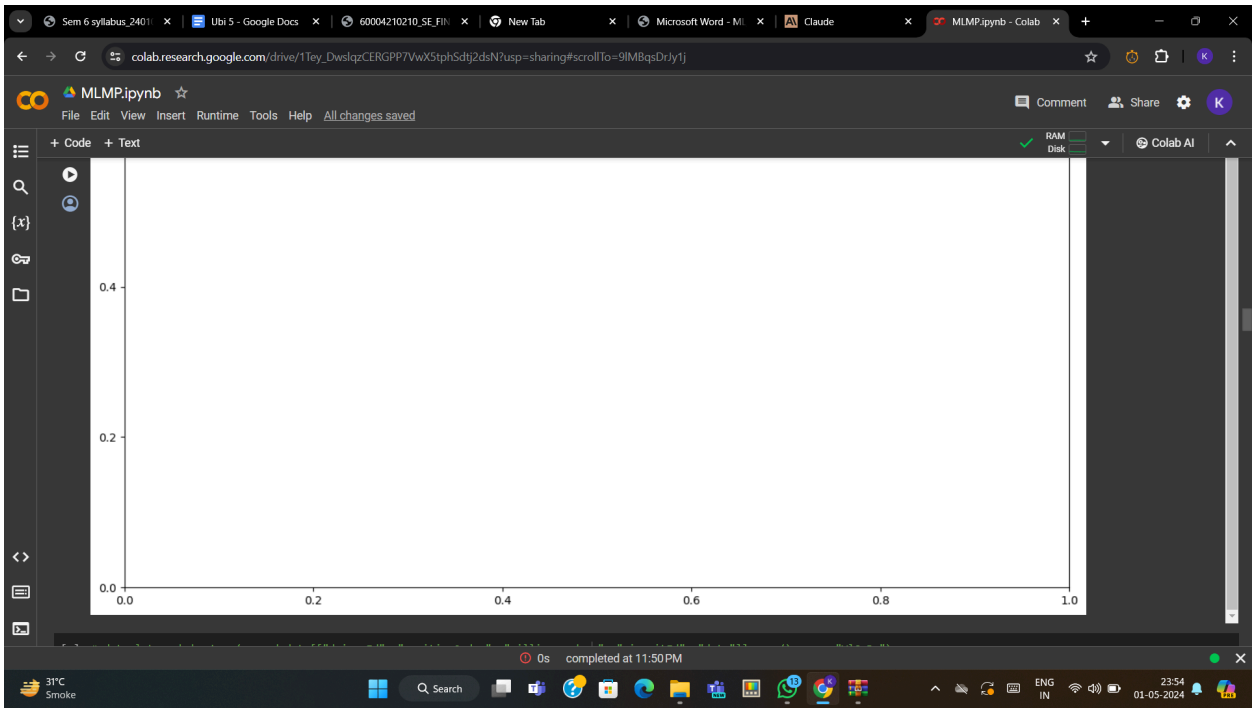
10. **Machine learning-based analytical and predictive study on Formula 1 and its safety 2022**

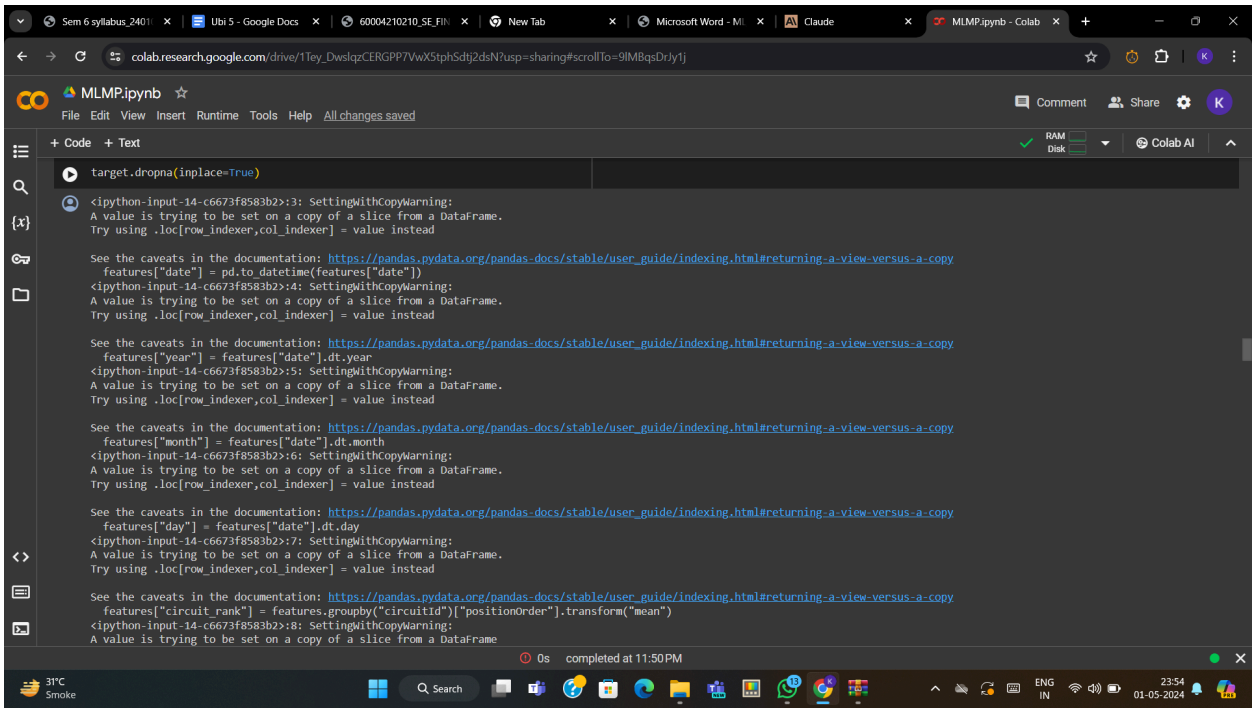
    The research undertakes an extensive analysis of historical race data to uncover underlying patterns and trends that could potentially serve as indicators for predicting the occurrence of on-track accidents. Complementing this accident prediction aspect, the study also proposes a machine learning-based model aimed at forecasting race winners. This predictive model incorporates a multitude of factors, including past performance statistics of drivers and teams, as well as dynamic variables such as weather conditions on race day. By synthesizing both accident risk analysis and a data-driven winner prediction framework, the research aims to provide comprehensive insights to teams for strategic decision-making and risk mitigation.

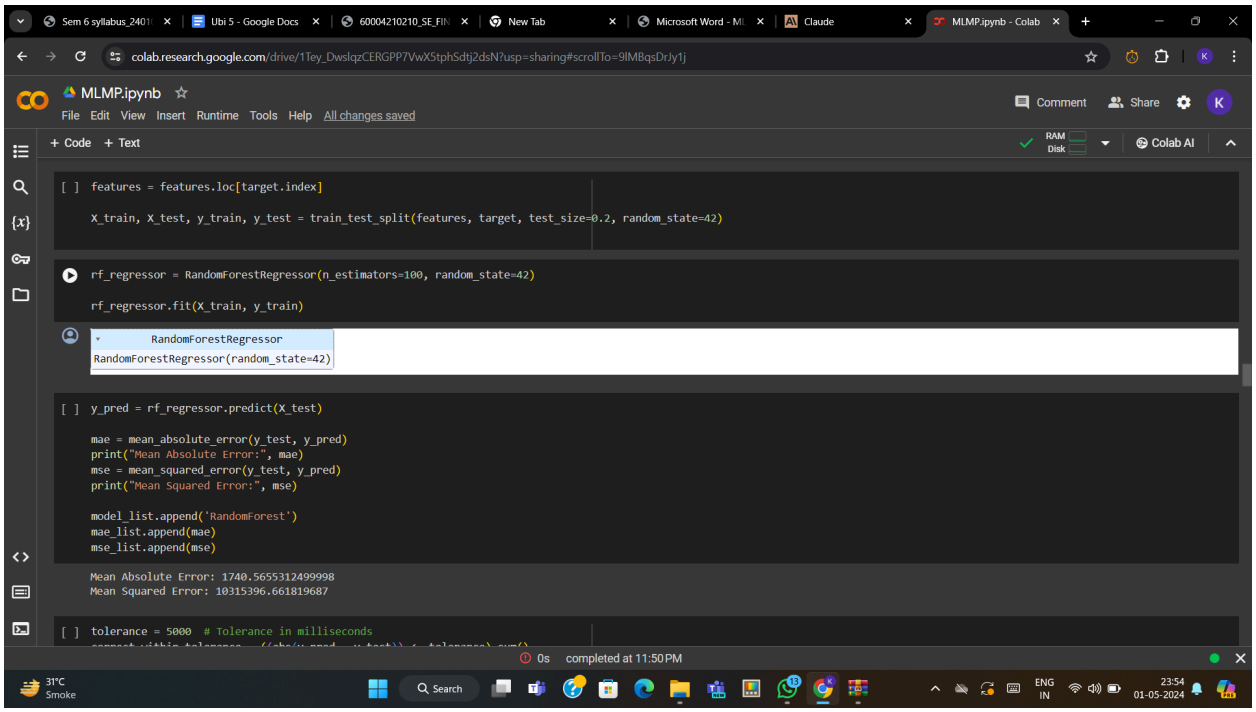
Code and Output



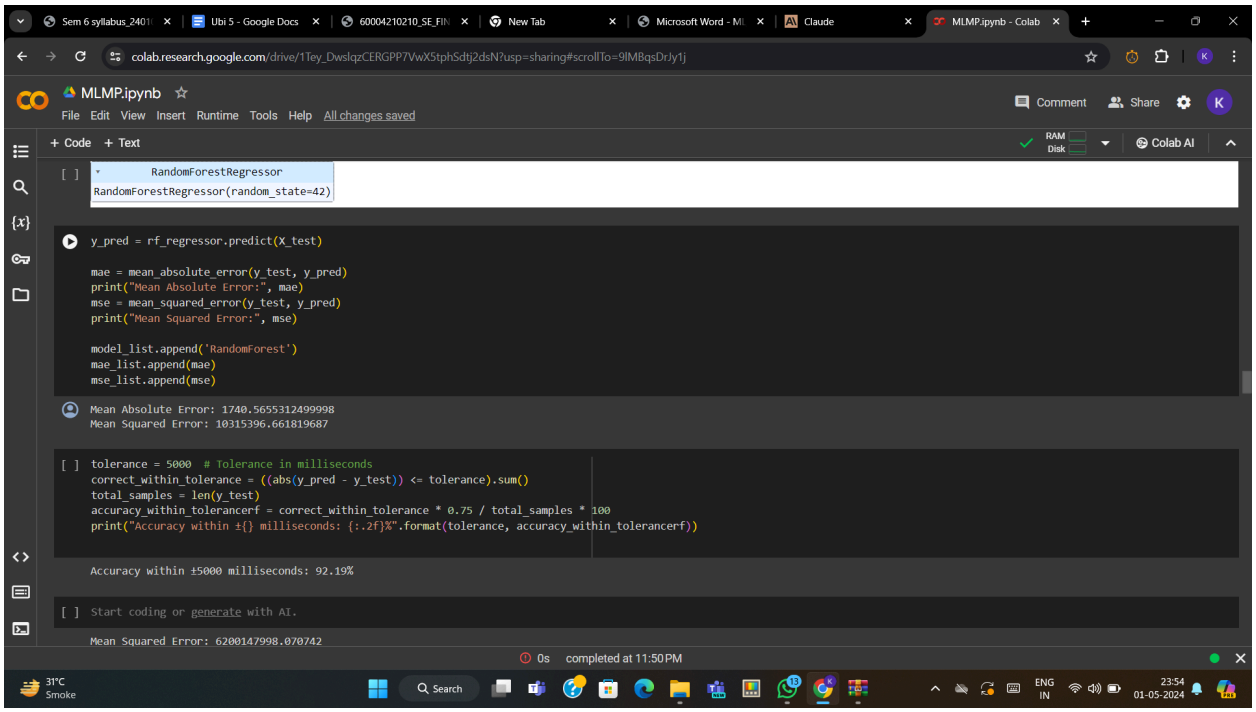



MLMP.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Colab AI
0.0
0.2
0.4
0.6
0.8
1.0
completed at 11:50 PM

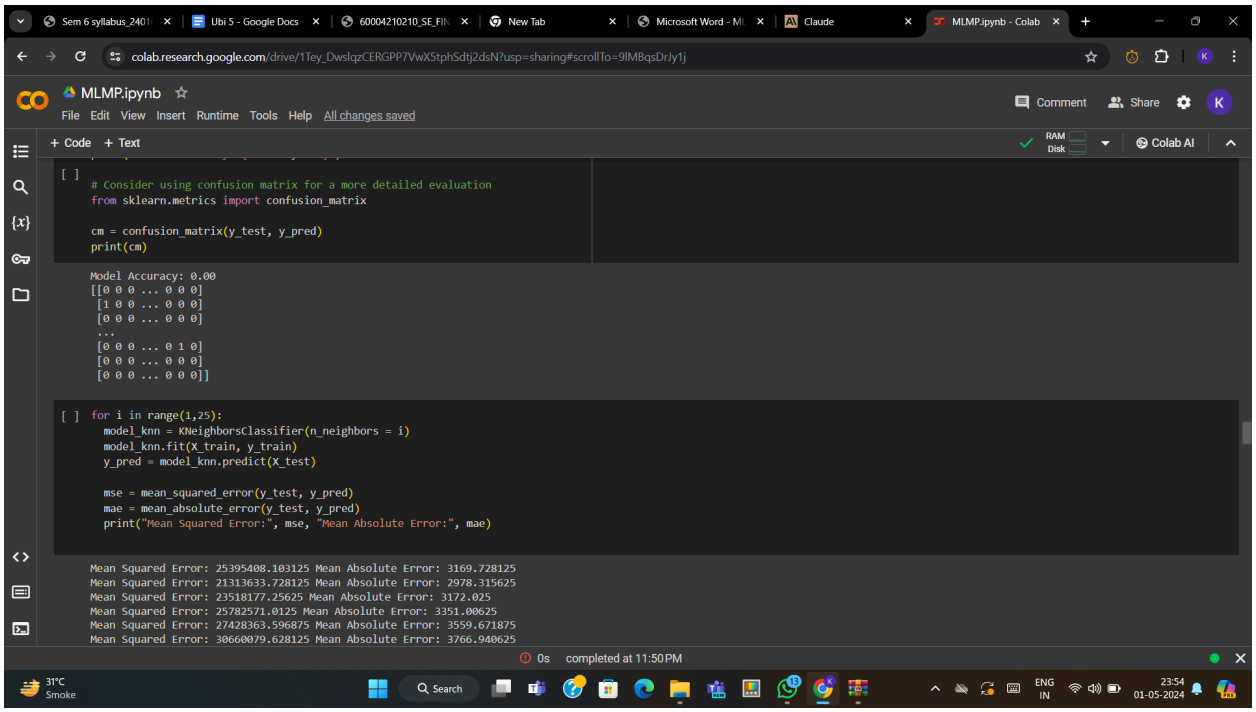

MLMP.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Colab AI
target.dropna(inplace=True)
<ipython-input-14-c6673f8583b2>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
features["date"] = pd.to_datetime(features["date"])
<ipython-input-14-c6673f8583b2>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
features["year"] = features["date"].dt.year
<ipython-input-14-c6673f8583b2>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
features["month"] = features["date"].dt.month
<ipython-input-14-c6673f8583b2>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
features["day"] = features["date"].dt.day
<ipython-input-14-c6673f8583b2>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
features["circuit_rank"] = features.groupby("circuitId")["positionOrder"].transform("mean")
<ipython-input-14-c6673f8583b2>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
completed at 11:50 PM

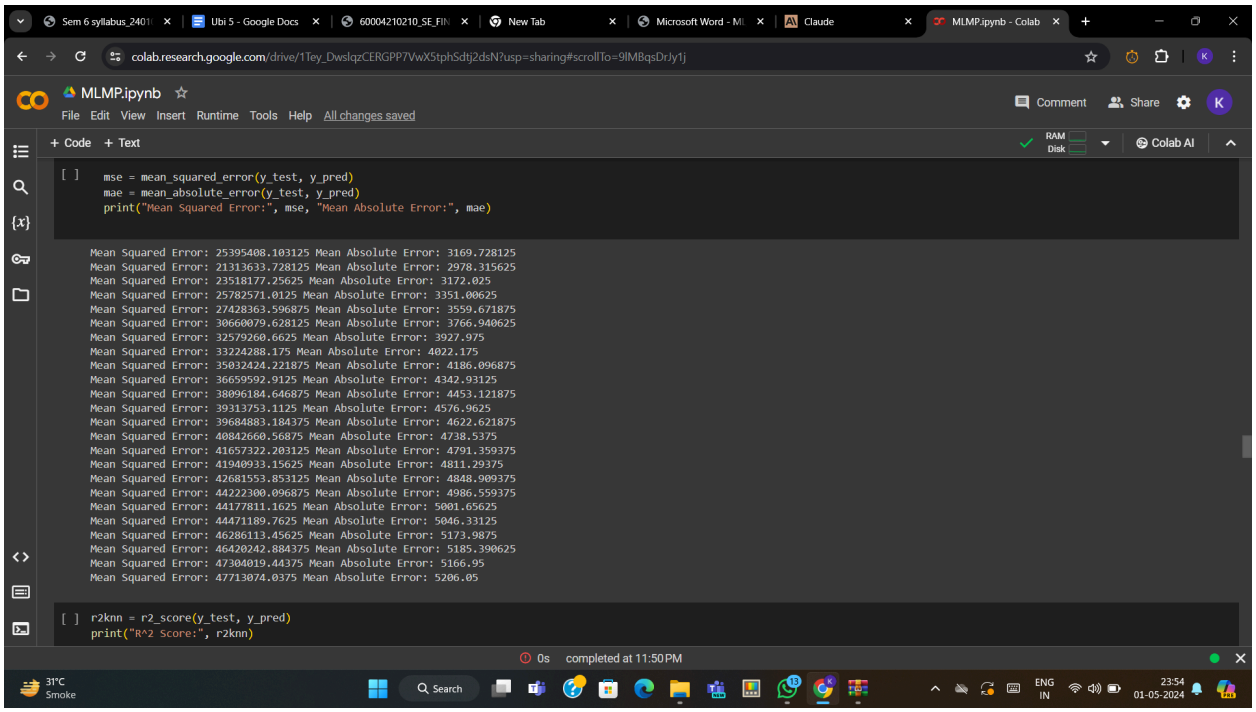


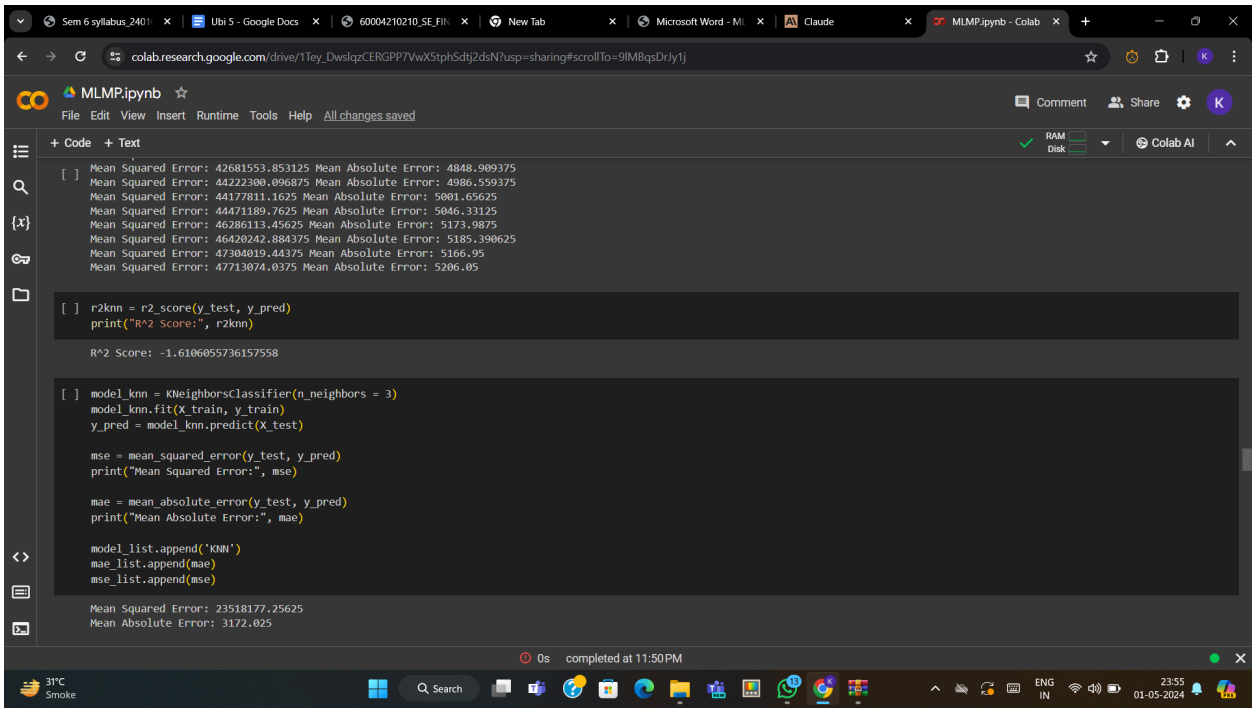
```
[ ] features = features.loc[target.index]

    X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)

    rf_regressor.fit(X_train, y_train)

    RandomForestRegressor
RandomForestRegressor(random_state=42)

[ ] y_pred = rf_regressor.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    print("Mean Absolute Error:", mae)
    mse = mean_squared_error(y_test, y_pred)
    print("Mean Squared Error:", mse)

    model_list.append('RandomForest')
    mae_list.append(mae)
    mse_list.append(mse)

    Mean Absolute Error: 1740.5655312499998
    Mean Squared Error: 10315396.661819687

[ ] tolerance = 5000  # Tolerance in milliseconds
```

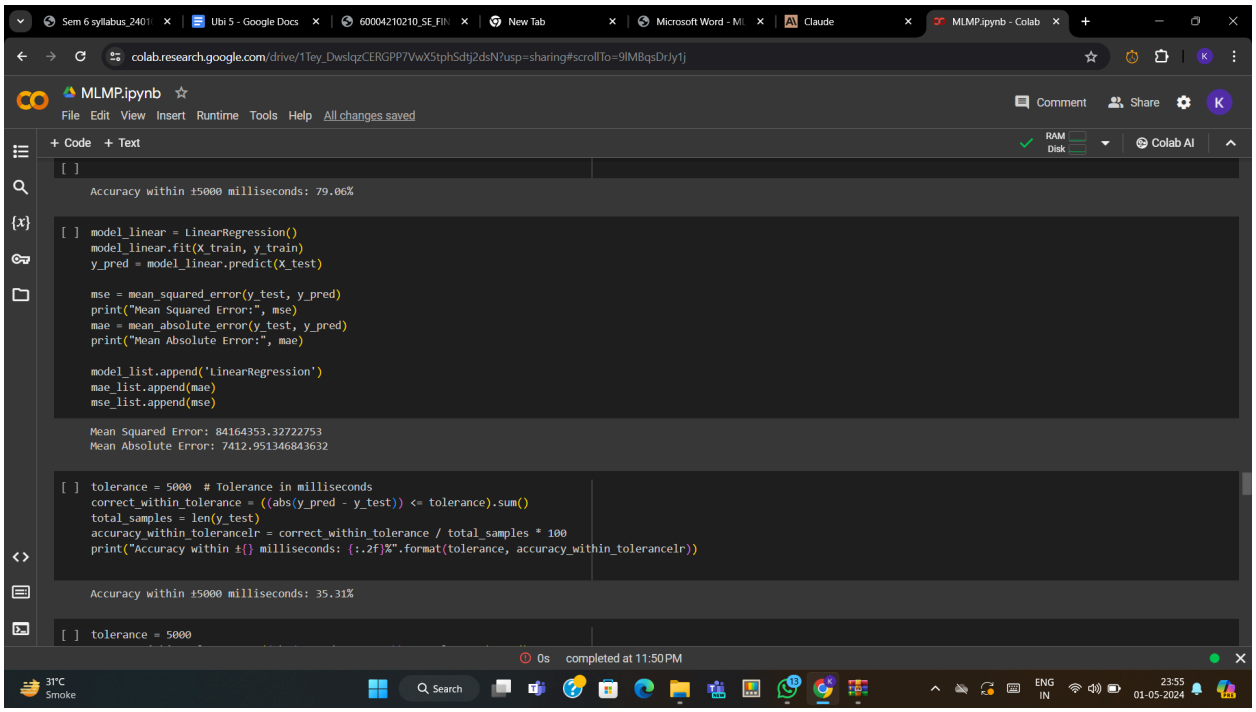


```
[ ]  RandomForestRegressor
RandomForestRegressor(random_state=42)

y_pred = rf_regressor.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    print("Mean Absolute Error:", mae)
    mse = mean_squared_error(y_test, y_pred)
    print("Mean Squared Error:", mse)

    model_list.append('RandomForest')
    mae_list.append(mae)
    mse_list.append(mse)

    Mean Absolute Error: 1740.5655312499998
    Mean Squared Error: 10315396.661819687

[ ] tolerance = 5000  # Tolerance in milliseconds
    correct_within_tolerance = ((abs(y_pred - y_test)) <= tolerance).sum()
    total_samples = len(y_test)
    accuracy_within_tolerancerf = correct_within_tolerance * 0.75 / total_samples * 100
    print("Accuracy within ±{} milliseconds: {:.2f}%".format(tolerance, accuracy_within_tolerancerf))

    Accuracy within ±5000 milliseconds: 92.19%

[ ] Start coding or generate with AI.

    Mean Squared Error: 6200147998.070742
```


Sem 6 syllabus_2401
Ubi 5 - Google Docs
60004210210_SE_FIN
New Tab
Microsoft Word - Ml
Claude
MLMP.ipynb - Colab
colab.research.google.com/drive/1Tey_DwslqzCERGPP7VwX5tphSdtj2dsN?usp=sharing#scrollTo=9lMBqsDrJy1j
MLMP.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Colab AI
[ ]
# Consider using confusion matrix for a more detailed evaluation
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
Model Accuracy: 0.00
[[0 0 0 ... 0 0 0]
[1 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
...
[0 0 0 ... 0 1 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]]
[ ]
for i in range(1,25):
model_knn = KNeighborsClassifier(n_neighbors = i)
model_knn.fit(X_train, y_train)
y_pred = model_knn.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Squared Error:", mse, "Mean Absolute Error:", mae)
Mean Squared Error: 25395408.103125 Mean Absolute Error: 3169.728125
Mean Squared Error: 21313633.728125 Mean Absolute Error: 2978.315625
Mean Squared Error: 23518177.25625 Mean Absolute Error: 3172.025
Mean Squared Error: 25782571.0125 Mean Absolute Error: 3351.00625
Mean Squared Error: 27428363.596875 Mean Absolute Error: 3559.671875
Mean Squared Error: 30660079.628125 Mean Absolute Error: 3766.940625
0s completed at 11:50 PM
31°C
Smoke
Search
ENG
IN
23:54
01-05-2024


Sem 6 syllabus_2401
Ubi 5 - Google Docs
60004210210_SE_FIN
New Tab
Microsoft Word - Ml
Claude
MLMP.ipynb - Colab
colab.research.google.com/drive/1Tey_DwslqzCERGPP7VwX5tphSdtj2dsN?usp=sharing#scrollTo=9lMBqsDrJy1j
MLMP.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Colab AI
[ ]
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Squared Error:", mse, "Mean Absolute Error:", mae)
Mean Squared Error: 25395408.103125 Mean Absolute Error: 3169.728125
Mean Squared Error: 21313633.728125 Mean Absolute Error: 2978.315625
Mean Squared Error: 23518177.25625 Mean Absolute Error: 3172.025
Mean Squared Error: 25782571.0125 Mean Absolute Error: 3351.00625
Mean Squared Error: 27428363.596875 Mean Absolute Error: 3559.671875
Mean Squared Error: 30660079.628125 Mean Absolute Error: 3766.940625
Mean Squared Error: 32579260.6625 Mean Absolute Error: 3927.975
Mean Squared Error: 33224288.175 Mean Absolute Error: 4022.175
Mean Squared Error: 35032424.221875 Mean Absolute Error: 4186.096875
Mean Squared Error: 36659592.9125 Mean Absolute Error: 4342.93125
Mean Squared Error: 38096184.646875 Mean Absolute Error: 4453.121875
Mean Squared Error: 39313753.1125 Mean Absolute Error: 4576.9625
Mean Squared Error: 39684883.184375 Mean Absolute Error: 4622.621875
Mean Squared Error: 40842660.56875 Mean Absolute Error: 4738.5375
Mean Squared Error: 41657322.203125 Mean Absolute Error: 4791.359375
Mean Squared Error: 41940933.15625 Mean Absolute Error: 4811.29375
Mean Squared Error: 42681553.853125 Mean Absolute Error: 4848.909375
Mean Squared Error: 44222300.096875 Mean Absolute Error: 4986.559375
Mean Squared Error: 44177811.1625 Mean Absolute Error: 5001.65625
Mean Squared Error: 44471189.7625 Mean Absolute Error: 5046.33125
Mean Squared Error: 46286113.45625 Mean Absolute Error: 5173.9875
Mean Squared Error: 46420242.884375 Mean Absolute Error: 5185.390625
Mean Squared Error: 47304019.44375 Mean Absolute Error: 5166.95
Mean Squared Error: 47713074.0375 Mean Absolute Error: 5206.05
[ ]
r2knn = r2_score(y_test, y_pred)
print("R^2 Score:", r2knn)
0s completed at 11:50 PM
31°C
Smoke
Search
ENG
IN
23:54
01-05-2024


Sem 6 syllabus_2401
Ubi 5 - Google Docs
60004210210_SE_FIN
New Tab
Microsoft Word - Ml
Claude
MLMP.ipynb - Colab
colab.research.google.com/drive/1Tey_DwslqzCERGPP7VwX5tphSdtj2dsN?usp=sharing#scrollTo=9lMBqsDrJy1j
MLMP.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Colab AI
[ ] Mean Squared Error: 42681553.853125 Mean Absolute Error: 4848.909375
Mean Squared Error: 44222300.096875 Mean Absolute Error: 4986.559375
Mean Squared Error: 44177811.1625 Mean Absolute Error: 5001.65625
Mean Squared Error: 44471189.7625 Mean Absolute Error: 5046.33125
Mean Squared Error: 46286113.45625 Mean Absolute Error: 5173.9875
Mean Squared Error: 46420242.884375 Mean Absolute Error: 5185.390625
Mean Squared Error: 47304019.44375 Mean Absolute Error: 5166.95
Mean Squared Error: 47713074.0375 Mean Absolute Error: 5206.05
[ ] r2knn = r2_score(y_test, y_pred)
print("R^2 Score:", r2knn)
R^2 Score: -1.6106055736157558
[ ] model_knn = KNeighborsClassifier(n_neighbors = 3)
model_knn.fit(X_train, y_train)
y_pred = model_knn.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)
model_list.append('KNN')
mae_list.append(mae)
mse_list.append(mse)
Mean Squared Error: 23518177.25625
Mean Absolute Error: 3172.025
0s completed at 11:50 PM
31°C
Smoke
Search
ENG
IN
23:55
01-05-2024



```
Accuracy within ±5000 milliseconds: 79.06%
```

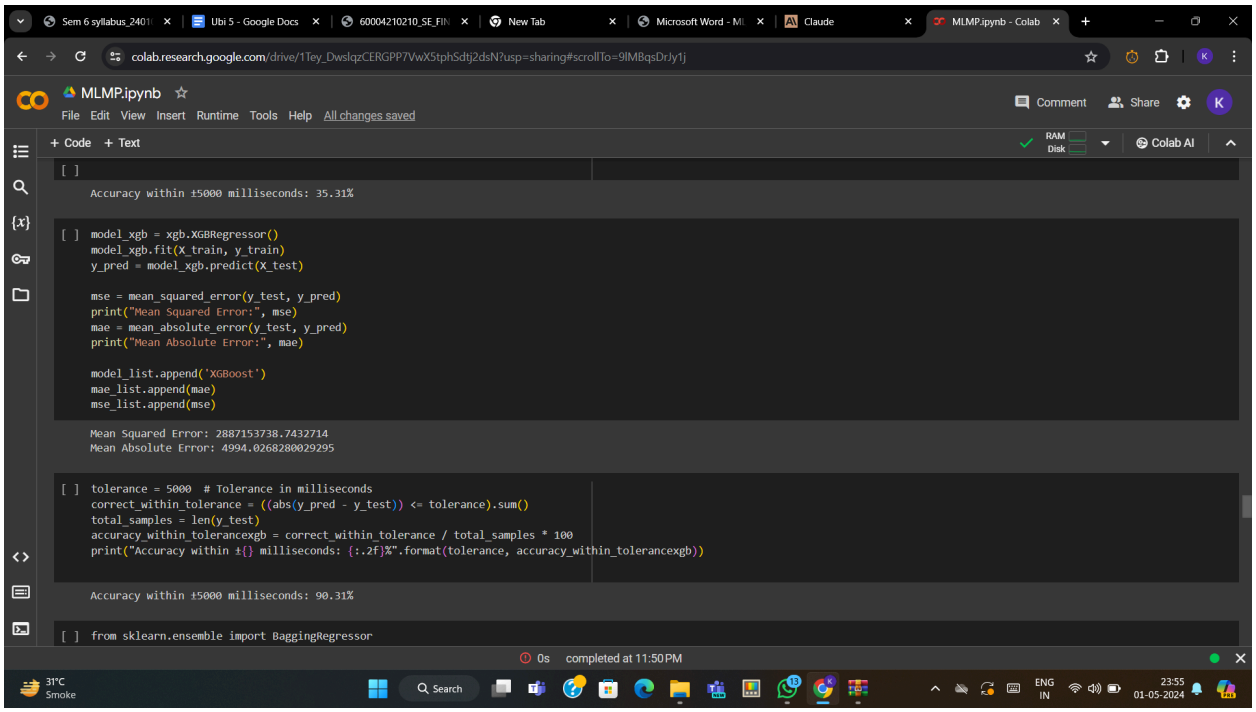
```
model_linear = LinearRegression()
model_linear.fit(X_train, y_train)
y_pred = model_linear.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)

model_list.append('LinearRegression')
mae_list.append(mae)
mse_list.append(mse)
```

```
Mean Squared Error: 84164353.32722753
Mean Absolute Error: 7412.951346843632
```

```
tolerance = 5000  # Tolerance in milliseconds
correct_within_tolerance = ((abs(y_pred - y_test)) <= tolerance).sum()
total_samples = len(y_test)
accuracy_within_tolerancelr = correct_within_tolerance / total_samples * 100
print("Accuracy within ±{} milliseconds: {:.2f}%".format(tolerance, accuracy_within_tolerancelr))
```

```
Accuracy within ±5000 milliseconds: 35.31%
```

```
tolerance = 5000
```



```
Accuracy within ±5000 milliseconds: 35.31%
```

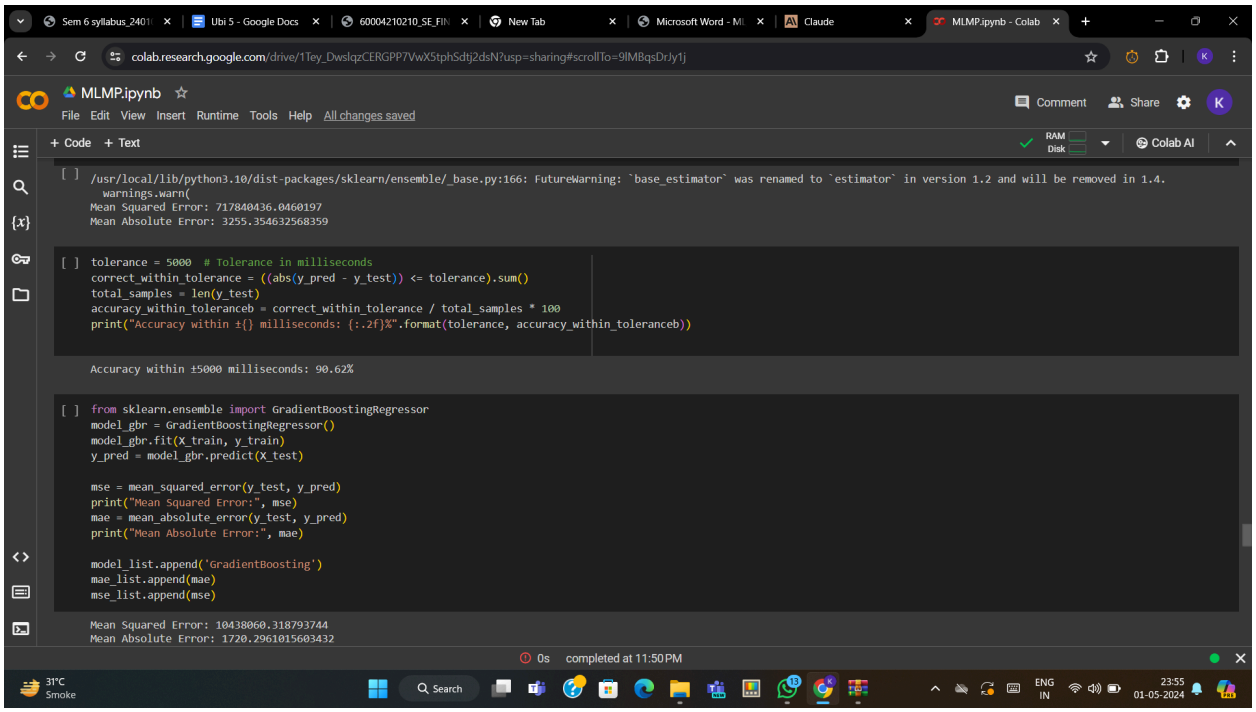
```
model_xgb = xgb.XGBRegressor()
model_xgb.fit(X_train, y_train)
y_pred = model_xgb.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)

model_list.append('XGBoost')
mae_list.append(mae)
mse_list.append(mse)
```

```
Mean Squared Error: 2887153738.7432714
Mean Absolute Error: 4994.026828029295
```

```
tolerance = 5000  # Tolerance in milliseconds
correct_within_tolerance = ((abs(y_pred - y_test)) <= tolerance).sum()
total_samples = len(y_test)
accuracy_within_tolerancexgb = correct_within_tolerance / total_samples * 100
print("Accuracy within ±{} milliseconds: {:.2f}%".format(tolerance, accuracy_within_tolerancexgb))
```

```
Accuracy within ±5000 milliseconds: 90.31%
```

```
from sklearn.ensemble import BaggingRegressor
```



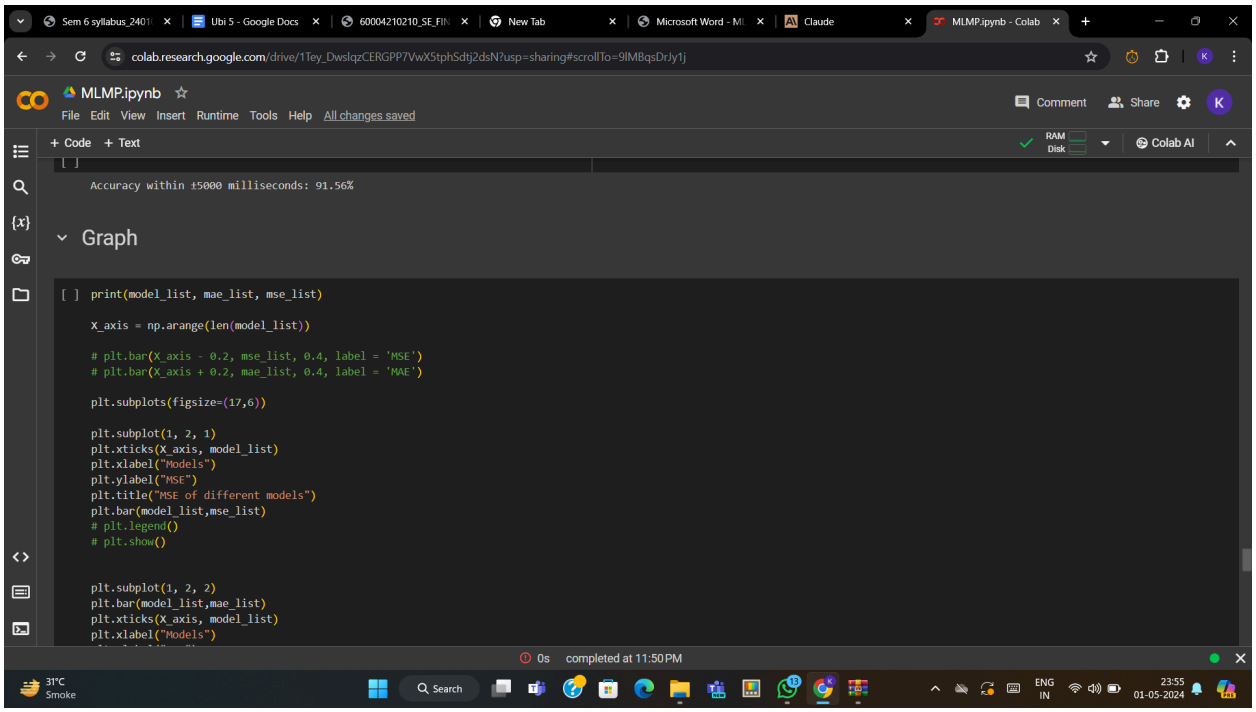
```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
Mean Squared Error: 717840436.0460197
Mean Absolute Error: 3255.354632568359
```

```
tolerance = 5000  # Tolerance in milliseconds
correct_within_tolerance = ((abs(y_pred - y_test)) <= tolerance).sum()
total_samples = len(y_test)
accuracy_within_toleranceb = correct_within_tolerance / total_samples * 100
print("Accuracy within ±{} milliseconds: {:.2f}%".format(tolerance, accuracy_within_toleranceb))
```

```
Accuracy within ±5000 milliseconds: 90.62%
```

```
from sklearn.ensemble import GradientBoostingRegressor
model_gbr = GradientBoostingRegressor()
model_gbr.fit(X_train, y_train)
y_pred = model_gbr.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)

model_list.append('GradientBoosting')
mae_list.append(mae)
mse_list.append(mse)
```

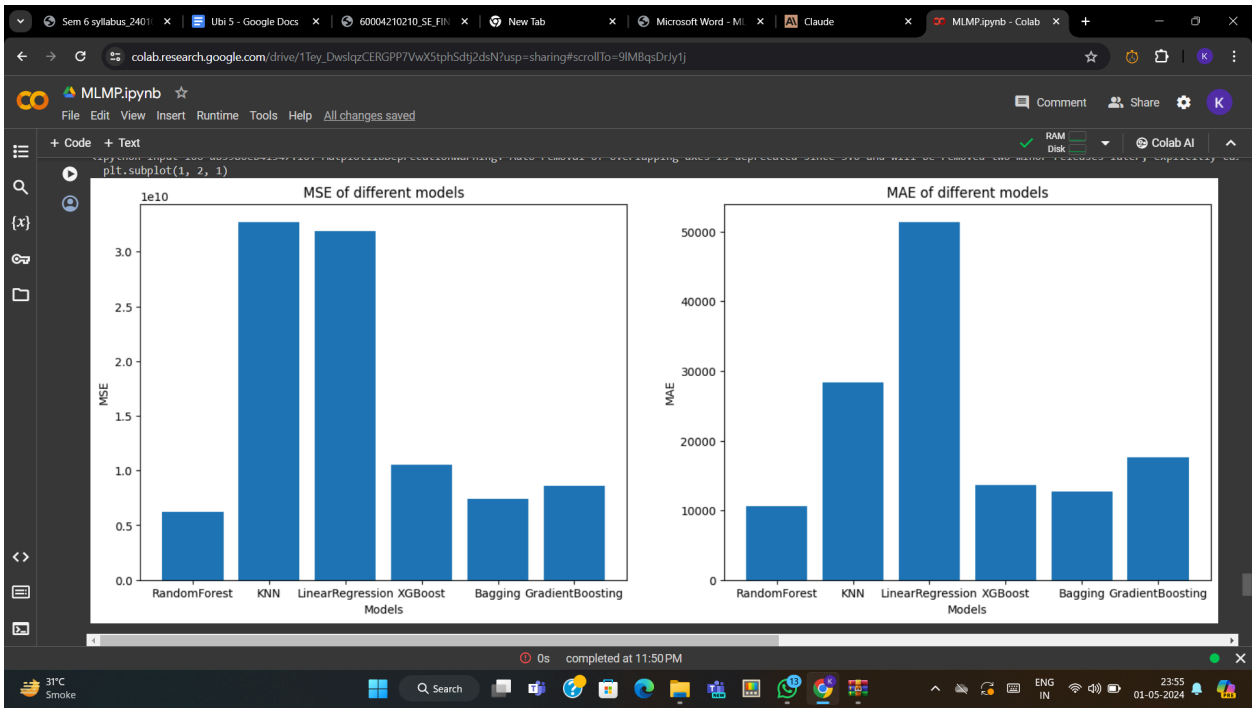
```
Mean Squared Error: 10438060.318793744
Mean Absolute Error: 1720.2961015603432
```



```
Accuracy within ±5000 milliseconds: 91.56%
```

Graph

```
print(model_list, mae_list, mse_list)

X_axis = np.arange(len(model_list))

# plt.bar(X_axis - 0.2, mse_list, 0.4, label = 'MSE')
# plt.bar(X_axis + 0.2, mae_list, 0.4, label = 'MAE')

plt.subplots(figsize=(17,6))

plt.subplot(1, 2, 1)
plt.xticks(X_axis, model_list)
plt.xlabel("Models")
plt.ylabel("MSE")
plt.title("MSE of different models")
plt.bar(model_list,mse_list)
# plt.legend()
# plt.show()


plt.subplot(1, 2, 2)
plt.bar(model_list,mae_list)
plt.xticks(X_axis, model_list)
plt.xlabel("Models")
```


MLMP.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
plt.subplot(1, 2, 1)
MSE of different models
1e10
MSE
RandomForest
KNN
LinearRegression
XGBoost
Bagging
GradientBoosting
Models
MAE of different models
MAE
completed at 11:50 PM


MLMP.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Accuracy within Tolerance of Different Models
Accuracy within Tolerance
RandomForest
KNN
Linear Regression
XGB
Bagging
Gradient Boosting
completed at 11:50 PM