David Galati University of Twente May 2024

AAIB Course – Assignment 2

Acknowledgement: ChatGPT was used both in the generation of text and code

Task 1:

Using methodological frameworks like CRISP-DM in machine learning (ML) projects brings substantial benefits that align with both technical execution and business objectives, as outlined in Lecture 2. CRISP-DM, a popular and systematic approach to data mining and ML, emphasizes a structured and iterative process, enhancing both project management and outcome quality.

Technical Benefits:	Business Benefits:
Structured Process: CRISP-DM provides a clear, step-by-step guide that includes business understanding, data understanding, data preparation, modeling, evaluation, and deployment. This structure helps ensure that critical steps are not overlooked, enhancing project coherence and completeness. **Reproducibility and Consistency:** Following a standardized framework like CRISP-DM ensures that processes can be replicated, which is vital for verifying results and improving future projects. The detailed and systematic approach of CRISP-DM supports consistent results across different projects and teams. **Enhanced Communication and Collaboration:** CRISP-DM fosters better communication among project stakeholders by providing a common language and expectations. This shared understanding facilitates collaboration between data scientists, engineers, and business stakeholders, ensuring that all parties are aligned with the project goals. **Risk Management:** By defining each phase clearly, potential risks such as data quality issues, model	**Alignment with Business Goals:** CRISP-DM starts with a thorough business understanding phase, ensuring that ML projects are closely aligned with business objectives. This alignment helps in developing solutions that deliver tangible business value, as seen in the Emons Group case study where AI was used to reduce empty mileage in logistics. **Efficient Resource Utilization:** The structured nature of CRISP-DM allows for better planning and resource allocation, optimizing time and costs. This efficiency is crucial for businesses looking to maximize return on investment from their ML initiatives. **Stakeholder Confidence and Trust:** Adopting a well-recognized framework like CRISP-DM enhances stakeholder confidence in the project's process and outcomes. The systematic approach and adherence to best practices provide assurance that the project is being handled professionally and effectively. **Scalability and Adaptability:** CRISP-DM's framework is versatile and can be adapted to various industries and project types. This scalability allows businesses to apply the same best practices across

performance challenges, and deployment problems can be identified and mitigated early. This proactive approach to risk management helps in maintaining project timelines and quality standards.	different projects, ensuring high standards and continuous improvement.

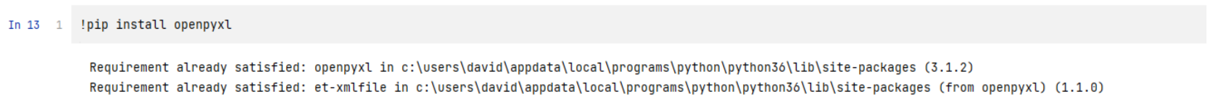
Task 2:

Explore the practical2 jupyter notebook, fix 5 existing errors (e.g., semantic, logical, runtime, etc.), and submit notebook without errors.

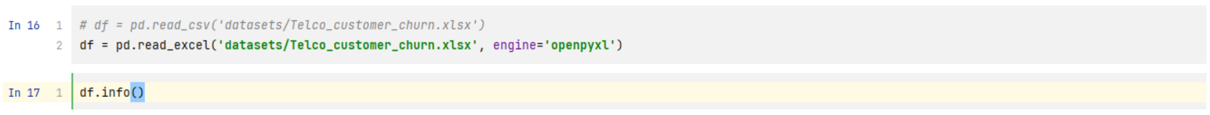
I have identified and fixed the following mistakes:

`read_csv()` instead of `read_excel()`:

Needed to install `openpyxl` to fix it:

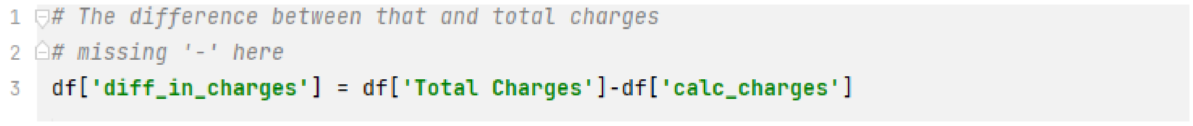


```
In 13  !pip install openpyxl

       Requirement already satisfied: openpyxl in c:\users\david\appdata\local\programs\python\python36\lib\site-packages (3.1.2)
       Requirement already satisfied: et-xmlfile in c:\users\david\appdata\local\programs\python\python36\lib\site-packages (from openpyxl) (1.1.0)
```



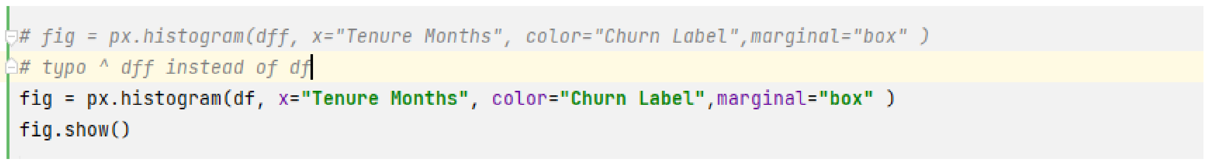
```
In 16  # df = pd.read_csv('datasets/Telco_customer_churn.xlsx')
       df = pd.read_excel('datasets/Telco_customer_churn.xlsx', engine='openpyxl')

In 17  df.info()
```

Minus sign missing here:



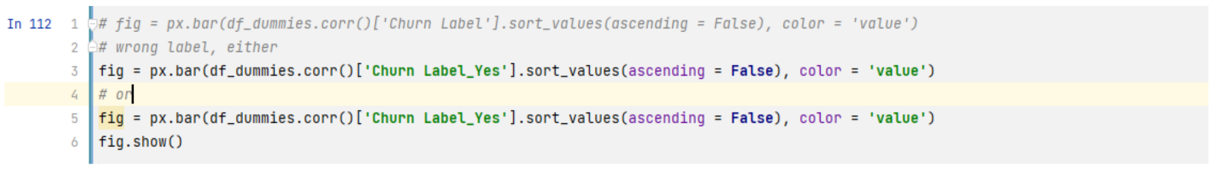
```
# The difference between that and total charges
# missing '-' here
df['diff_in_charges'] = df['Total Charges']-df['calc_charges']
```

Typo here:



```
# fig = px.histogram(dff, x="Tenure Months", color="Churn Label",marginal="box" )
# typo ^ dff instead of df
fig = px.histogram(df, x="Tenure Months", color="Churn Label",marginal="box" )
fig.show()
```

Wrong label here:



```
In 112  # fig = px.bar(df_dummies.corr()['Churn Label'].sort_values(ascending = False), color = 'value')
        # wrong label, either
        fig = px.bar(df_dummies.corr()['Churn Label_Yes'].sort_values(ascending = False), color = 'value')
        # or
        fig = px.bar(df_dummies.corr()['Churn Label_Yes'].sort_values(ascending = False), color = 'value')
        fig.show()
```

The working Jupyter notebook is packaged alongside this report.

Reflect on the notebook structure: Steps on data preprocessing, model implementation. What three top things you would do differently (better)? List what would you do differently and explain why.

Modularize the Notebook

One significant improvement would be to modularize the code by splitting it into distinct, reusable functions. This involves separating tasks such as data loading, preprocessing, encoding, scaling, and visualization into individual functions. Modularization enhances readability by organizing the notebook into clearly defined sections. It also facilitates debugging and maintenance, as issues can be isolated within specific functions. Furthermore, reusable functions enable code reuse across different projects, promoting efficiency and consistency in data processing workflows.

Improve Error Handling and Data Validation

Another key improvement would be to incorporate robust error handling and data validation steps. This ensures that the dataset is correctly formatted and contains all necessary columns before proceeding with analysis or model training. By implementing checks for required columns and handling missing or incorrect data gracefully, the notebook becomes more resilient to unexpected data issues. This validation step prevents runtime errors that could disrupt the analysis and ensures that the data integrity is maintained throughout the workflow.

Enhance Data Visualization

Improving the data visualization aspect of the notebook is also crucial. Many of the current visualizations lack descriptive titles, axis labels, and contextual information, which can make them harder to interpret. Adding these elements to each plot will provide better insights and make the findings more accessible to the audience. Additionally, using consistent color schemes and formatting across visualizations will enhance the overall aesthetic and readability of the notebook. Effective visualizations are key to communicating insights clearly and can significantly impact the understanding of the analysis results.

Task 3:

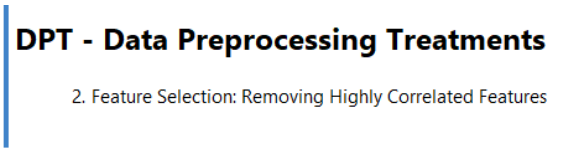
In Lecture 1 (practical1), we did some Data Preprocessing Treatment (DPT) and used Linear Regression and Decision Tree Regressor algorithms to predict housing prices. In this task, you should:

• Add some DPT tasks (at least two different treatments not addressed in the lecture) that might boost predictions (e.g., Feature Engineering, Correlation, etc., etc.)

Interaction features can capture relationships between existing features that may not be obvious when considered in isolation. These features can provide additional predictive power to your models.

```
# 1. Feature Engineering: Creating Interaction Features

housing['rooms_per_household'] = housing['total_rooms'] / housing['households']
housing['bedrooms_per_room'] = housing['total_bedrooms'] / housing['total_rooms']
housing['population_per_household'] = housing['population'] / housing['households']
```

Removing highly correlated features can help reduce multicollinearity, which can improve the stability and performance of machine learning models.



DPT - Data Preprocessing Treatments

2. Feature Selection: Removing Highly Correlated Features

```
# Correlation matrix
corr_matrix = housing.corr()

# Remove highly correlated features
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
high_correlation = [column for column in upper.columns if any(upper[column] > 0.9)]

housing = housing.drop(columns=high_correlation)

```

• Implement other ML algorithms - At least 3 ML regressor-based algortihms (e.g., Random Forest, Gradient Boosting, Adaboost, Support Vector Machines, Neural Networks, LightGBM, etc., etc.), and compare performance (error).

Three machine learning regressor models were used, the results from the three machine learning regressor models—Random Forest Regressor, Gradient Boosting Regressor, and

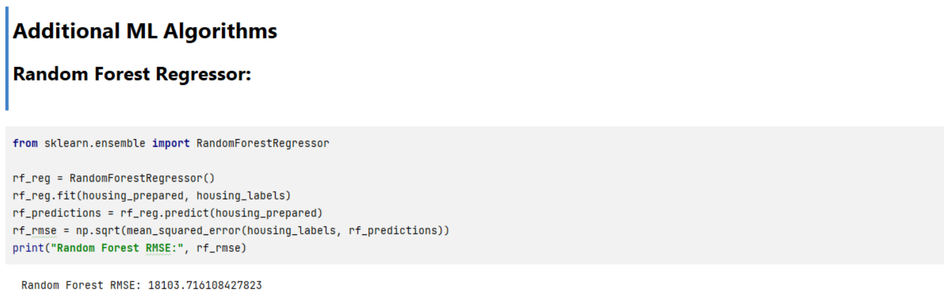
Support Vector Regressor (SVR)—show significant differences in their predictive performance, as measured by the Root Mean Squared Error (RMSE).

The Random Forest Regressor model achieved the lowest RMSE of 18013.716108427823, indicating the best performance among the three models. Random Forest's ability to reduce overfitting by averaging multiple decision trees and its capability to capture complex interactions between features likely contributed to this superior performance.

The Gradient Boosting model produced an RMSE of 52490.611030410706, which is significantly higher than that of the Random Forest Regressor but much lower than the SVR. Gradient Boosting works by building trees sequentially, where each new tree corrects the errors of the previous ones. Despite being powerful, it can be sensitive to hyperparameter settings and requires careful tuning, which may explain its relatively higher RMSE in this context.

The SVR model exhibited the highest RMSE of 118250.6374191385, indicating the least effective performance in this scenario. SVR, while robust to high-dimensional spaces and effective in certain contexts, might not have been as well-suited to this dataset's characteristics or required more extensive hyperparameter optimization to perform competitively.

In summary, the Random Forest Regressor outperformed the other two models, making it the best choice for predicting housing prices in this dataset. The Gradient Boosting Regressor, though less accurate than Random Forest, still showed potential with further tuning. The SVR, however, demonstrated the need for significant improvements to be competitive in this regression task.



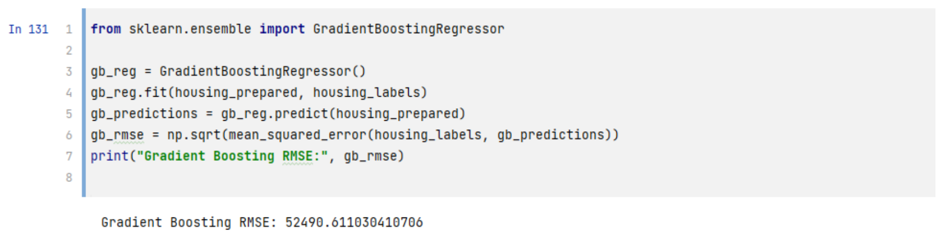
Additional ML Algorithms

Random Forest Regressor:

```
from sklearn.ensemble import RandomForestRegressor

rf_reg = RandomForestRegressor()
rf_reg.fit(housing_prepared, housing_labels)
rf_predictions = rf_reg.predict(housing_prepared)
rf_rmse = np.sqrt(mean_squared_error(housing_labels, rf_predictions))
print("Random Forest RMSE:", rf_rmse)
```

```
Random Forest RMSE: 18103.716108427823
```

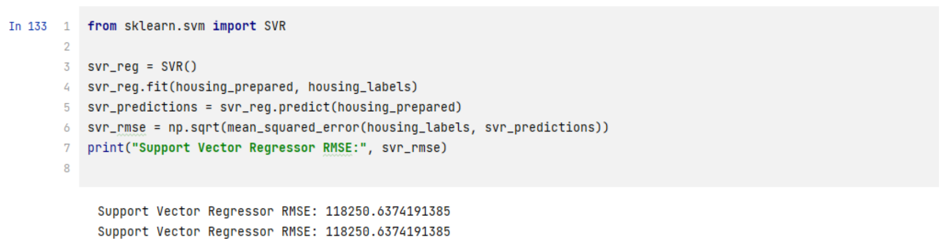
Figure 1 Random Forest



```
from sklearn.ensemble import GradientBoostingRegressor

gb_reg = GradientBoostingRegressor()
gb_reg.fit(housing_prepared, housing_labels)
gb_predictions = gb_reg.predict(housing_prepared)
gb_rmse = np.sqrt(mean_squared_error(housing_labels, gb_predictions))
print("Gradient Boosting RMSE:", gb_rmse)

```

```
Gradient Boosting RMSE: 52490.611030410706
```

Figure 2 Gradient Boosting



```
In 133
from sklearn.svm import SVR

svr_reg = SVR()
svr_reg.fit(housing_prepared, housing_labels)
svr_predictions = svr_reg.predict(housing_prepared)
svr_rmse = np.sqrt(mean_squared_error(housing_labels, svr_predictions))
print("Support Vector Regressor RMSE:", svr_rmse)
```

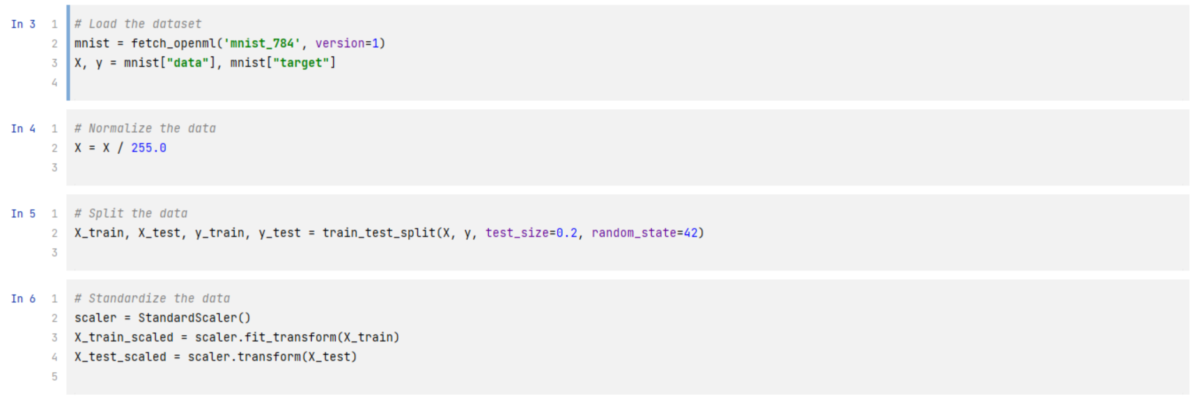
```
Support Vector Regressor RMSE: 118250.6374191385
Support Vector Regressor RMSE: 118250.6374191385
```

Figure 3 SVR Regressor

In Lecture 1 (practical1), we showed the "Hello World!" problem in ML: Hand-digit identification. In this task, you must:

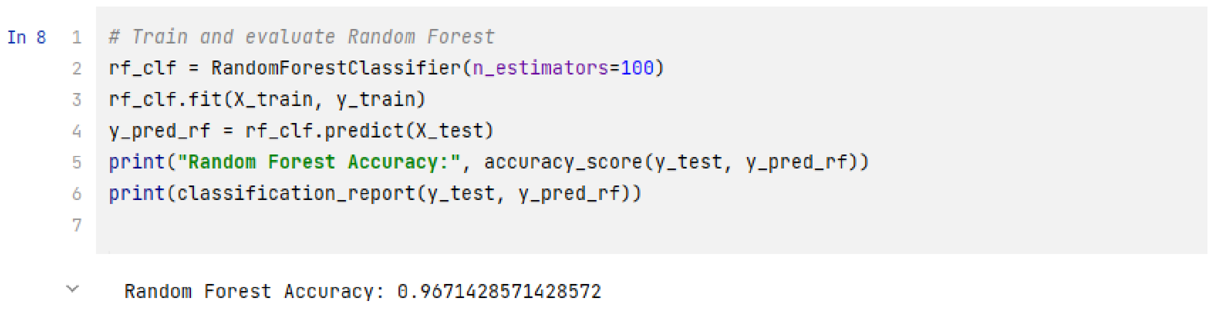
Use ChatGPT or Gemini (you have to create an account) or use any other open-source LLM to create an ML pipeline for you and the coding you need (using at least 3 ML classifiers). Then, based on the tools' output, answer: what would you do differently (or do on top of what was provided)?

Full code in hand_digit_notebook.ipynb



```
In 3
# Load the dataset
mnist = fetch_openml('mnist_784', version=1)
X, y = mnist["data"], mnist["target"]

In 4
# Normalize the data
X = X / 255.0

In 5
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

In 6
# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Figure 4 Used fetch_openml to get the mnist_784 dataset. Did train-test split, standardized the data



```
In 8
# Train and evaluate Random Forest
rf_clf = RandomForestClassifier(n_estimators=100)
rf_clf.fit(X_train, y_train)
y_pred_rf = rf_clf.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.9671428571428572
```

Figure 5 Used a Random Forest Classifier and checked it's accuracy.

The following things could be done differently to improve the solution:

Data Augmentation:

Implement data augmentation techniques to artificially expand the training dataset by creating modified versions of images, such as rotations, translations, and scaling. This can help improve the robustness of the models.

Hyperparameter Tuning:

Use techniques like Grid Search or Randomized Search for hyperparameter tuning to find the optimal settings for each model, potentially improving their performance.

Advanced Feature Extraction:

Use Principal Component Analysis (PCA) or other dimensionality reduction techniques to reduce the feature space and improve model training efficiency.

Ensemble Methods:

Combine predictions from multiple models to create an ensemble, potentially increasing overall accuracy and robustness. Techniques like voting classifiers or stacking can be used.

Cross-Validation:

Implement k-fold cross-validation to ensure that the models' performance is consistent and not dependent on a single train-test split.

Additional Performance Metrics:

Evaluate models using additional metrics like precision, recall, F1-score, and confusion matrix to get a more comprehensive understanding of their performance.

I have tried implementing some of those things. See `improved_hand_digit_notebook.ipyn`

b