Project 4 - Group 1

1. Objectives

The idea is to build a model to predict the price of Toronto real estate listings and classify their price compared to the market. The project will exist after class as a tool for personal use when we want to participate in the Toronto real estate market, and possibly as a website (pending data permissions).

2. Data Collection

- Home Price Index Data
- Toronto Real Estate Board (TREB) PDFs: Monthly data going back to 1996. Extract tables from PDFs using libraries tabula-py or camelot-py. Part of preprocessing could also involve adjusting varying structures of different PDFs if they changed through time.
- Kaggle Dataset: It's three years old, and the amount of data might be limited.
- Listing.ca: this has tons of monthly data going back to 2014 which we could scrape, if we wanted to get more granular with the types of homes (# of bedrooms, bathrooms, etc.) we could also add filters when scraping to get that.
- Canadian Housing Market Stats: also from CREA.
- If the website allows scraping (check website's robots.txt file or terms of service), use web scraping libraries like BeautifulSoup to gather additional data.
- Plan B in case Canadian data doesn't come through: Zillow

3. Data Cleaning and Exploration

- Clean the dataset by handling missing values, outliers, and duplicates.
- Visualization tools like Matplotlib or Seaborn can be helpful.
- Use PDF extraction library (as mentioned above) to convert the tables in TREB Monthly PDFs into a structured format (example: CSV or DataFrame).
- Combine the data extracted from PDFs with the scraped data, if needed.
- Perform necessary cleaning and preprocessing steps to create a unified dataset.
- Suggestions: PostgreSQL for data storage. Seems like Apache Spark is another viable option.
- Not sure if this is needed but I just watched this video that mentioned Spark SQL and Spark Streaming that handles real-time data processing. Just wanted to note it here as possible options for this project under Apache Spark.

4. Features

Select relevant features for the prediction model: location, property size, number of bedrooms, and any other relevant information. Create new features or transform existing ones to improve predictive performance, example: price per square foot or property age.

5. **Data Preprocessing**

- Convert variables into a format suitable for machine learning models (one-hot encoding, label encoding).
- Standardize or normalize numerical features.
- Handling any remaining missing values.

6. **Model Selection**

Machine learning model for regression (to predict prices) and classification (to classify prices compared to the market). Examples: Linear Regression, Random Forest, or Gradient Boosting for regression, and Logistic Regression or Decision Trees for classification. *NOTE:* It seems like Random Forests can handle both regression and classification simultaneously.

Example using random forest regressor:
https://nbviewer.org/github/srnghn/ml_example_notebooks/blob/master/Predicting%20Yacht%20Resistance%20with%20Decision%20Trees%20%26%20Random%20Forests.ipynb

Classification might need "sold" data, so maybe that part won't be ML.

Cross Value Scores applied:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

Cross Value Scores needed pipeline to work:
https://www.geeksforgeeks.org/make_pipeline-function-in-sklearn/

7. **Model Training**

Split the dataset into training and testing sets.
Train models using the training set.
Implement early-stopping to prevent over-fitting.
NOTE: If the DataFrame is not too big we could consider Cache/Persist for training purposes. Just a suggestion in case we can use it.

8. **Model Evaluation**

Evaluate models:

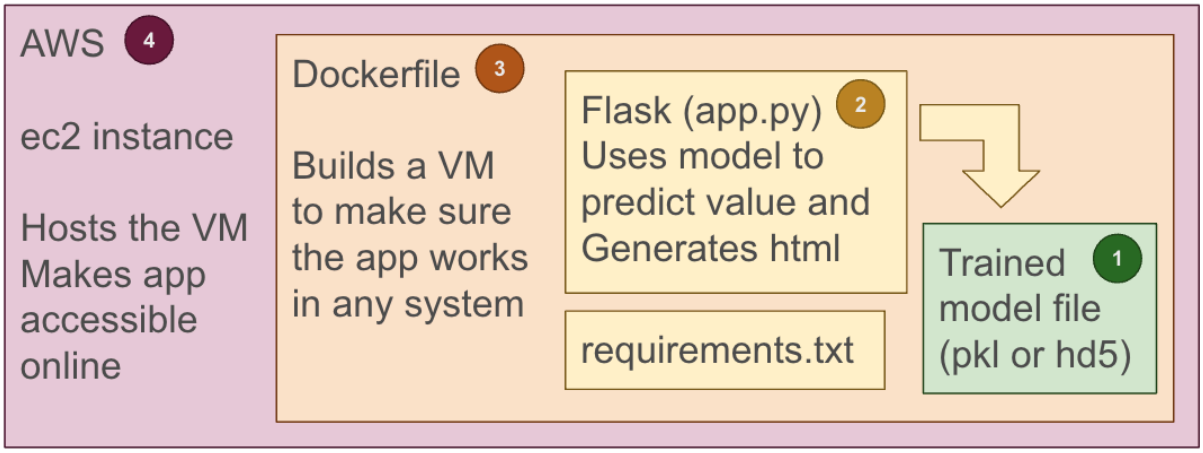
- Regression (Mean Squared Error).
- Classification (Accuracy, Precision, Recall, F1 Score)
- Requirements ask for a minimum of: 75% accuracy and/or 0.80 R-squared.

9. **Hyperparameter Tuning**

Adjust hyperparameters of the models to improve performance (we could potentially use Fine tuning here?)

10. **Deployment**

Deployment Tools: Flask (just a suggestion)
—
Steps for deploying (based on blog below):



1. After training and fine-tuning: Save model object (pickle or hd5 file)
2. Flask app that includes:
    - Loading model object once, when the app starts
    - Routes:
        - "Main" is the landing page, this is where users will interact with the app. The user will input the test data here, press a button that will call the "predict" route and receive an answer that will be displayed.
        - "Predict" takes in test data (i.e. the listing we want to predict), uses the model object to predict a price and returns that price. The blog does this with a POST method
        - Not sure how but it also needs to take that number and compare it with the actual price in the listing
3. Use docker to "containerize" which makes sure the app works in any system by running it in a virtual machine (docker image). It generates a Dockerfile that uses:
    - requirements.txt includes all the libraries we are using and their versions
    - Flask app
    - Model file (saved as pickle or hd5, read by the app)
4. EC2 instance for hosting the docker container (virtual machine) and making it reachable by the outside world. For this, we need to:
    - Create a pem file with a key pair, and make its permissions private
    - Launch an ec2 instance

        - Select the Amazon Instance Machine (what os to use)
        - Select the instance type (hardware)
        - Edit security groups to allow http traffic on port 80
        - When prompted, give it the name of the pem file with the key pair
        - Wait for its status to turn to “running”
    - From the local terminal, ssh (log in) into the ec2 instance (we will need the pem file):
        - Install docker inside the instance
        - Exit and log in again
        - Check if docker works
        - Log out again
        - From the local terminal (not from within ec2): Copy the files we need to build the docker image: requirements.txt, app.py, trained model file and Dockerfile. Then check that copying was successful
        - Build and run the docker image using the exact same commands that were used in the local system (with port 80)
        - Check that it worked by using the browser to hit the different routes

—

Blog: Simple was to deploy machine learning models : The steps in this blog post are very comprehensive for our project. In this example they only used EC2 Instances from AWS to deploy their model, but I will provide insights into all AWS tools in case we find something else useful, or necessary (as we go). Below I highlighted some of the ones I think we don’t need at all, but maybe we could benefit from EC2, Lambda, EMR and Sagemaker (just suggestions). AWS Free Tier Account has services that could support our project like:

- EC2 Instances: For hosting backend services, machine learning models, and Flask.
    - EC2 provides scalable compute capacity in the cloud. It allows users to run virtual servers (instances) to host their applications.
    - Users can use EC2 instances to host the backend services, deploy machine learning models, and run their Flask application.
- S3: For storing and managing large datasets. (I don’t think we will need this one at all but including it just in case).
    - S3 is a scalable object storage service designed to store and retrieve any amount of data from anywhere on the web.
    - Use S3 for storing and managing large datasets related to Toronto real estate listings. We can upload, download, and organize datasets in S3 buckets.
- RDS (Relational Database Service): For PostgreSQL. (I don’t think we will need this one at all but including it just in case).
    - RDS is a managed relational database service that makes it easy to set up, operate, and scale a relational database in the cloud.
    - We could use RDS with PostgreSQL to store structured data related to real estate listings. RDS manages routine database tasks such as backups, software patching, and scaling, allowing users to focus on their application logic.

- Lambda: For serverless functions or automation tasks.
    - Lambda is a serverless computing service that runs code in response to events and automatically manages the computing resources.
    - Users can use Lambda for serverless functions or automation tasks. For example, they might use Lambda to trigger a function whenever new data is added to their S3 bucket, allowing for automated data processing or model updates without the need for managing server infrastructure.
- Spark on EMR (Elastic MapReduce): For large-scale data processing using Apache Spark.
    - EMR is a cloud-based big data platform that simplifies processing and analysis of large datasets using popular frameworks such as Apache Spark and Apache Hadoop.
    - Users can use EMR with Spark to perform large-scale data processing tasks. EMR clusters can be easily provisioned, scaled, and terminated based on processing needs. It's suitable for tasks like data cleaning, transformation, and analysis.
- Sagemaker: For machine learning model training and deployment.
    - Sagemaker is a fully managed service that enables users to quickly build, train, and deploy machine learning models at scale.
    - We can use Sagemaker for machine learning model training and deployment. It simplifies the end-to-end machine learning workflow, providing tools for data preparation, model training, and deployment. Models can be deployed as endpoints for predictions.
- API Gateway: For creating APIs to interact with machine learning models. (I don't think we will need this one at all but including it just in case).
    - API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.
    - We can use API Gateway to create APIs that interact with the machine learning models.
- AWS Free Tier offers: 12 months free, example: 750 hours of EC2 usage per month, each tool will have their own usage per month.

Other than AWS we could use:

- Google Cloud Platform: they also have a free tier.
- Microsoft Azure: free tier available as well.
- IBM Cloud: it seems like these services are just always free on their free tier.
- DigitalOcean: offers a free tier as well.
- From what I can see all of these services have similar tools to AWS so we should be covered regardless of the option we choose.

11. Continuous Improvement

Keep refining the model based on new data and feedback. We can consider incorporating more advanced techniques or exploring deep learning if needed, in the future.

Challenges with availability of historical data and future improvements could include more precise predictions based on actual sales data, instead of only current listings.
Different Property Types have unique features that must be accounted for, in the future, as we are capable of accessing more data, we could create separate models for each Property Type for more accurate predictions.

NOTES:
- Cloud ETL is mentioned in the bonus part but it seems like there would most likely be a cost associated with creating a Cloud ETL model.