Player Valuation Predictions

Made by Mohammad Haaris Imtiaz, Davud Ciftci & Abdul Kadir Seker

Intro

- Problem & Motivation
- Approach
  - Data
  - Feature Selection
  - NN training & LightGBM
  - Hyper parameter tuning
- Solution
- Possible improvements

Problem & Motivation

- Problem: How can we predict player valuation based on passed data?
- Importance of the problem:
  - Transfers
  - Scouting
  - Academy
- Motivation:
  - Enhanced resource distribution
  - Key qualities
  - Possibly one step ahead of the market

Data

- Source
  - Transfermarkt
- Csv’s
  - players
  - player_valuations
  - appearances
  - game_events

- 60,000+ games from many seasons on all major competitions
- 400+ clubs from those competitions
- 30,000+ players from those clubs
- 400,000+ player market valuations historical records
- 1,200,000+ player appearance records from all games

Feature selection

- Initial (Made new features from the raw data, such as total appearances, total goals, total assists, and player age)

- Manual selection(We also chose features that were important for us manually)

- N-Computed feature selection(Used a Gradient Boosting Regressor (XGBoost) and SelectFromModel with a 'mean' threshold to select the most important features.)

Feature selection

We trained an XGBoost Regressor to determine the importance of the initial features (including one-hot encoded categorical features).

We then used SelectFromModel with a threshold='mean' to select features whose importance was greater than the average feature importance.

Initially, using the 'median' threshold kept all features (468). Switching to the 'mean' threshold reduced the number of selected features to 69. These selected features included the engineered statistical features and several one-hot encoded categorical features (sub-position, position, foot, and specific clubs).

Feature selection

Total features before selection: 468

Selected features: 69

- height_in_cm
- total_appearances
- total_yellow_cards
- total_red_cards
- total_minutes_played
- total_goals
- total_assists
- age
- sub_position_Central Midfield
- sub_position_Left Midfield
- position_Attack
- position_Defender
- position_Midfield
- foot_both
- Rest are major football clubs

NN training & LightGBM Regressor(First attempt):

We trained two different regression models:

LightGBM Regressor(First attempt):

A gradient boosting model known for its speed and efficiency.

Trained on a split of the training data (80% train_part, 20% validation).

Used early_stopping based on the validation loss (l1 or MAE) to prevent overfitting. The model stopped after 675 boosting rounds.

Neural Network(First attempt):

A simple sequential model with Dense layers, Batch Normalization, and Dropout.

Trained using the Adam optimizer and Mean Squared Error (MSE) loss.

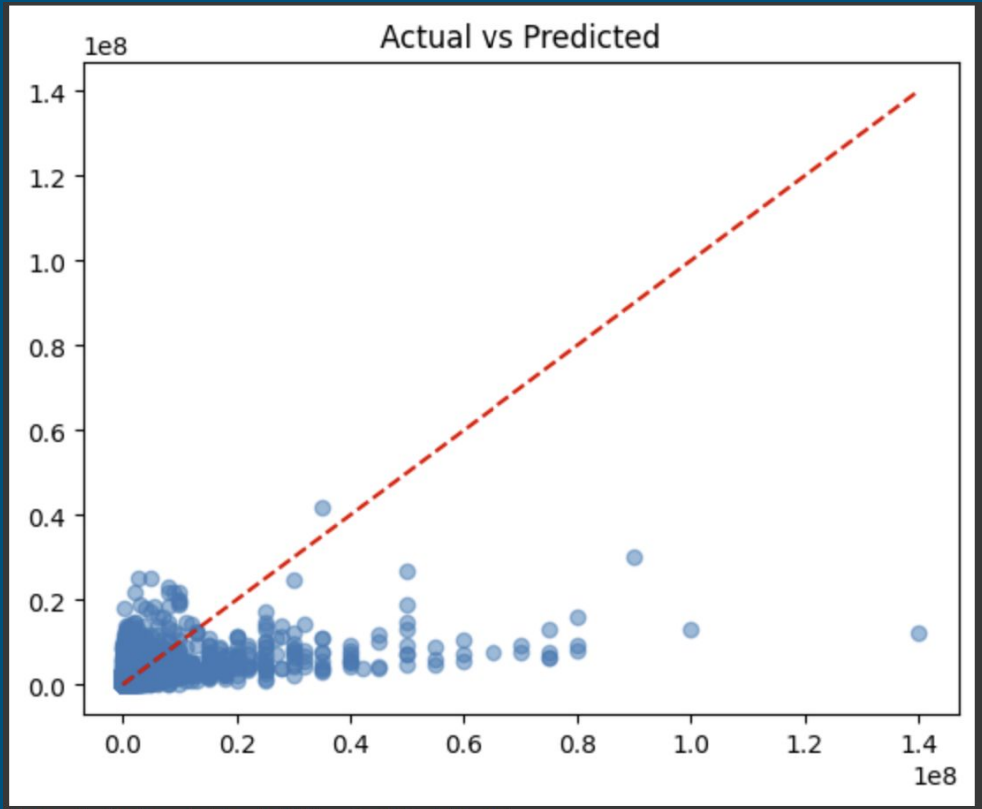
Used EarlyStopping and ReduceLROnPlateau callbacks to manage training and learning rate.

Both models were trained on the selected features (X_selected) and the target variable (y).

NN training & LightGBM Regressor(Second attempt):

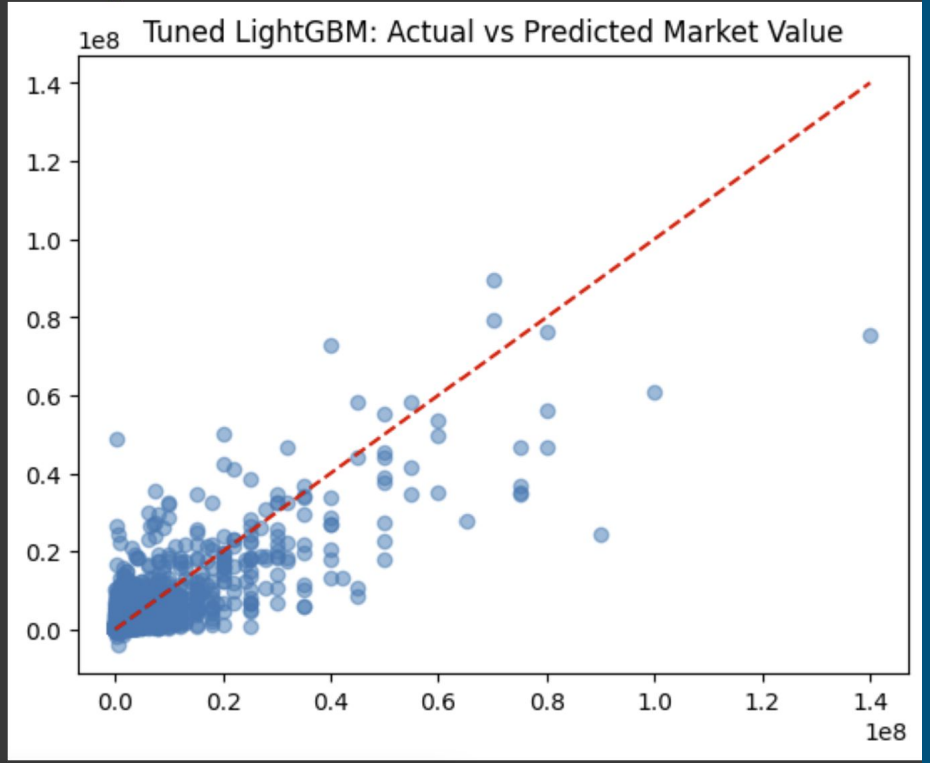
- A deeper `Sequential` model with Dense layers, Batch Normalization, and Dropout to improve generalization and training stability.
- Architecture: 128 → 64 → 32 → 1 with ReLU activations and Dropout (0.3, 0.2).
- Trained with Adam optimizer (`lr=1e-3`) and MSE loss.
- Used `EarlyStopping` and `ReduceLROnPlateau` to prevent overfitting and adapt the learning rate.
- Trained for up to 200 epochs with batch size 64 and 20% validation split.
- Used a GPU-accelerated LightGBM regressor for faster training.
- Tuned hyperparameters using `RandomizedSearchCV` over 50 combinations with 3-fold CV.
- Evaluated using negative MAE; training was parallelized across CPU cores.
- Enabled GPU training with `device_type='gpu'` and optimized using `max_bin=63`.

NN(First attempt)

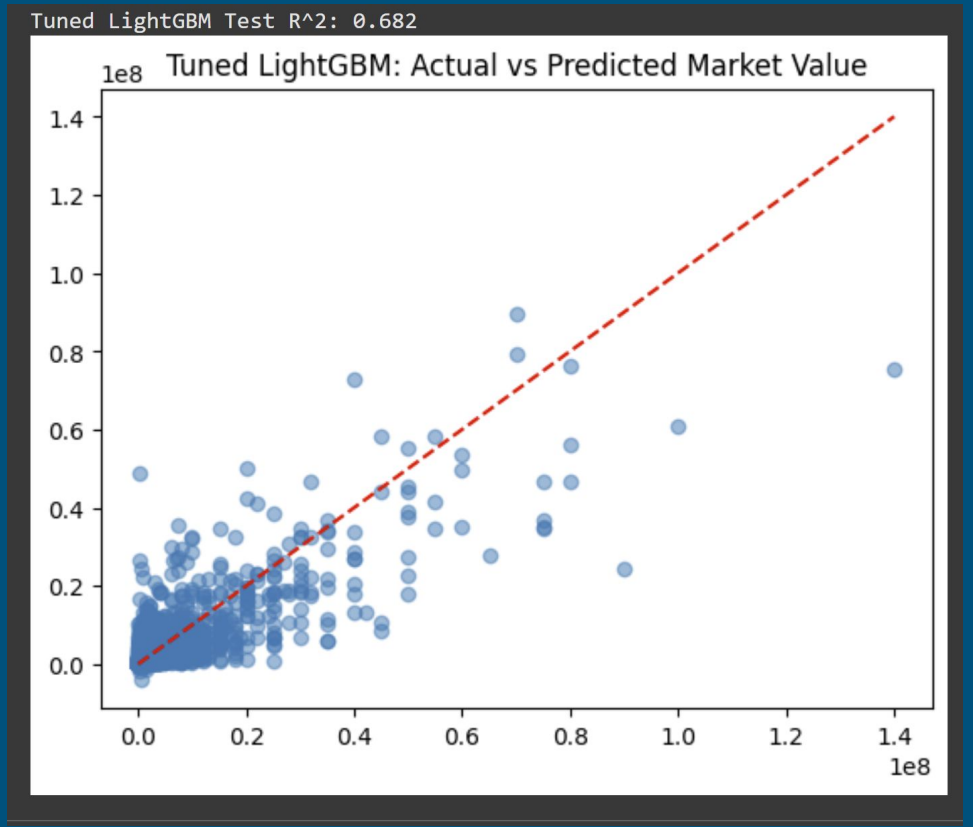


LightGBM(First attempt)

```
Tuned LightGBM Test MAE: 1,139,878.87
Tuned LightGBM Test R^2: 0.682
```



LightGBM(Second attempt)



Hyperparameter tuning

Performed a RandomizedSearchCV to optimize the hyperparameters of the LightGBM model.

Approach: Tuned hyperparameters using `RandomizedSearchCV`, sampling 50 combinations from a defined search space.

Validation: Used 3-fold cross-validation to evaluate model performance.

Objective: Minimized Mean Absolute Error (MAE) across folds.

Key Tuned Parameters:

- `n_estimators`, `learning_rate`, `num_leaves`
- `reg_alpha`, `reg_lambda` (regularization)
- `colsample_bytree`, `subsample` (sampling strategies)
- `min_child_samples` (minimum data in leaf)

Results for the best hyperparamters:

'subsample': 0.9,

'reg_lambda': 0.5,

'reg_alpha': 0.0,

'num_leaves': 31,

'n_estimators': 1000,

'min_child_smples': 10,

'learning_rate': 0.01,

'colsample_bytree': 1.0

Results for the LightGBM

- **Initial LightGBM Model (before tuning):**
  - Test MAE: €1,241,436.02
  - Test R^2: 0.621
- **Tuned LightGBM Model (after RandomizedSearchCV):**
  - Test MAE: €1,139,878.87
  - Test R^2: 0.682

- **Our interpretation:**
  - The tuned LightGBM model showed an improvement in R-squared compared to the initial model, indicating that hyperparameter tuning helped capture more variance in the market value.
  - While the R-squared of 0.682 for the tuned LightGBM model indicates a reasonably good fit, there is still room for improvement.

Evaluation

Assessed the performance of the models using metrics like Mean Absolute Error (MAE) and R-squared (R^2).

Possible improvements

Further work could involve more extensive hyperparameter tuning, trying more advanced models, or engineering more sophisticated features.

Appendix

```
from google.colab import userdata
import os

os.environ["KAGGLE_KEY"] = userdata.get('KAGGLE_KEY')
os.environ["KAGGLE_USERNAME"] = userdata.get('KAGGLE_USERNAME')
```

```
!kaggle datasets download -d davidcariboo/player-scores
```

```
! mkdir -p data
! unzip player-scores.zip -d data
```

```
import os
from pathlib import Path
import pandas as pd
import numpy as np
from functools import reduce
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import matplotlib.pyplot as plt

# --- Paths ---
DATA_DIR = "data"
```

```
# --- Data Loading ---
def load_csv(name: str) -> pd.DataFrame:
    """
    Load a CSV file from the data directory by name (without extension).
    Example: load_csv('players') reads 'data/players.csv'.
    """
    path = "data/" + f"{name}.csv"
    print(path)
    return pd.read_csv(path)
```

```
# Convenience functions for each file
def load_appearances(): return load_csv('appearances')
```

```
def load_club_games(): return load_csv('club_games')
```

```
def load_clubs(): return load_csv('clubs')
```

```
def load_competitions():     return load_csv('competitions')
```

```
def load_game_events(): return load_csv('game_events')


def load_game_lineups(): return load_csv('game_lineups')


def load_games(): return load_csv('games')


def load_player_valuations(): return load_csv('player_valuations')


def load_players(): return load_csv('players')


def load_transfers(): return load_csv('transfers')


def load_all() -> dict[str, pd.DataFrame]:
    """
    Load every .csv file in your data directory into a dict of DataFrames.
    Keys are the filename without extension.
    """
    data = {}
    for fp in Path(DATA_DIR).glob("*.csv"):
        key = fp.stem
        data[key] = pd.read_csv(fp)
    return data


# --- Merge Helpers ---
def merge_two(df1: pd.DataFrame, df2: pd.DataFrame, on: str, how: str = 'left') ->
    """Merge two DataFrames on a key."""
    return df1.merge(df2, on=on, how=how)


def merge_multiple(dfs: list, on: str, how: str = 'left') -> pd.DataFrame:
    """Sequentially merge a list of DataFrames on a key."""
    return reduce(lambda left, right: left.merge(right, on=on, how=how), dfs)


# --- Feature Engineering ---
# ------------------------------------------------------------------
#  INDIVIDUAL PLAYER-LEVEL ADDERS
#  Each helper → adds ONE column and returns a NEW dataframe
# ------------------------------------------------------------------

# ---------- appearances-based ----------
def add_total_appearances(players: pd.DataFrame,
                          appearances: pd.DataFrame,
                          colname: str = "total_appearances") -> pd.DataFrame:
    """Count distinct games per player and add `total_appearances`."""
    app_counts = (
        appearances
```

```
        appearances
        .groupby("player_id")["game_id"]
        .nunique()
        .rename(colname)
        .reset_index()
    )
    return players.merge(app_counts, on="player_id", how="left")


def add_total_yellow_cards(players: pd.DataFrame,
                           appearances: pd.DataFrame,
                           colname: str = "total_yellow_cards") -> pd.DataFrame:
    yellows = (
        appearances
        .groupby("player_id")["yellow_cards"]
        .sum(min_count=1)
        .rename(colname)
        .reset_index()
    )
    return players.merge(yellows, on="player_id", how="left")


def add_total_red_cards(players: pd.DataFrame,
                        appearances: pd.DataFrame,
                        colname: str = "total_red_cards") -> pd.DataFrame:
    reds = (
        appearances
        .groupby("player_id")["red_cards"]
        .sum(min_count=1)
        .rename(colname)
        .reset_index()
    )
    return players.merge(reds, on="player_id", how="left")


def add_total_minutes_played(players: pd.DataFrame,
                             appearances: pd.DataFrame,
                             colname: str = "total_minutes_played") -> pd.DataFrame
    if "minutes_played" in appearances.columns:
        minute_col = "minutes_played"
    elif "minutes" in appearances.columns:
        minute_col = "minutes"
    else:
        # fallback: assume 90’ for every appearance
        appearances = appearances.copy()
        appearances["assumed_minutes"] = 90
        minute_col = "assumed_minutes"

    mins = (
        appearances
        .groupby("player_id")[minute_col]
        .sum(min_count=1)
        .rename(colname)
        .reset_index()
    )
```

```
    )
    return players.merge(mins, on="player_id", how="left")


import pandas as pd

def add_total_GA(players: pd.DataFrame,
                 game_events: pd.DataFrame,
                 goal_col: str = "total_goals",
                 assist_col: str = "total_assists") -> pd.DataFrame:
    """Count total goals and total assists per player from game events."""
    # Count goals: every event with type 'Goals'
    goals = (
        game_events[game_events['type'] == 'Goals']
        .groupby('player_id')
        .size()
        .rename(goal_col)
        .reset_index()
    )

    # Count assists: every assist recorded in 'player_assist_id' for goal events
    assists = (
        game_events[game_events['type'] == 'Goals']
        .dropna(subset=['player_assist_id'])
        .groupby('player_assist_id')
        .size()
        .rename(assist_col)
        .reset_index()
        .rename(columns={'player_assist_id': 'player_id'})
    )

    # Merge counts into players dataframe
    players = (
        players
        .merge(goals, on='player_id', how='left')
        .merge(assists, on='player_id', how='left')
    )

    # Fill missing values with 0 and ensure integer type
    players[goal_col] = players[goal_col].fillna(0).astype(int)
    players[assist_col] = players[assist_col].fillna(0).astype(int)

    return players
# ---------- age ----------
def add_age(players: pd.DataFrame,
            date_col: str = "date_of_birth",
            ref_date: str | None = "2022-08-01",
            colname: str = "age") -> pd.DataFrame:
    """Compute age in whole years on `ref_date` (defaults to ‘today’)."""
    df = players.copy()
    birth = pd.to_datetime(df[date_col], errors="coerce")
    today = pd.Timestamp(ref_date) if ref_date else pd.Timestamp.today()

    years = today.year - birth.dt.year
    had_bday = (today.month > birth.dt.month) | (
```

```
    had_bday = (today.month > birth.dt.month) | (
        (today.month == birth.dt.month) & (today.day >= birth.dt.day)
    )
    df[colname] = (years - (~had_bday)).astype("Int64")
    return df



def get_latest_valuation(vals: pd.DataFrame) -> pd.DataFrame:
    vals['date'] = pd.to_datetime(vals['date'])
    idx = vals.groupby('player_id')['date'].idxmax()
    return vals.loc[idx, ['player_id', 'market_value_in_eur']].reset_index(drop=Tru


def merge_player_data(players, stats, latest_valuations):
    """
    Joins players with their aggregated stats and latest market valuation.
    """
    df = players.merge(stats, on='player_id', how='left')
    df = df.merge(latest_valuations[['player_id', 'market_value_in_eur']], on='play
    return df


# --- Preprocessing ---
def fillna_and_scale(df: pd.DataFrame, cols: list) -> tuple:
    df = df.copy()
    df[cols] = df[cols].fillna(0)
    scaler = StandardScaler()
    return scaler.fit_transform(df[cols]), scaler


def encode_categorical(df: pd.DataFrame, cols: list) -> tuple:
    encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
    arr = encoder.fit_transform(df[cols])
    df_enc = pd.DataFrame(arr, columns=encoder.get_feature_names_out(cols), index=d
    return df_enc, encoder


# --- Plotting ---
def plot_distribution(vals, title='Distribution', bins=50):
    plt.hist(vals, bins=bins)
    plt.title(title)
    plt.show()


def scatter_actual_vs_pred(y_true, y_pred, title='Actual vs Predicted'):
    plt.scatter(y_true, y_pred, alpha=0.5)
    plt.plot([y_true.min(), y_true.max()], [y_true.min(), y_true.max()], 'r--')
    plt.title(title)
    plt.show()


# --- Age Computation ---
def compute_age(df, date_col = 'date_of_birth', ref_date = '2022-08-01'):
    df_copy = df.copy()
```

```
    df_copy = df.copy()
    # Parse birth dates, coercing invalid entries to NaT
    birth = pd.to_datetime(df[date_col], errors='coerce')
    # Determine reference date
    if ref_date is None:
        today = pd.Timestamp.today()
    else:
        today = pd.to_datetime(ref_date)
    # Calculate year difference
    years = today.year - birth.dt.year
    # Determine if each birthday has occurred this year
    had_birthday = (today.month > birth.dt.month) | \
                   ((today.month == birth.dt.month) & (today.day >= birth.dt.day))
    # Compute age, subtracting 1 where birthday hasn't occurred
    age = years - (~had_birthday).astype(pd.Int64Dtype())
    # Ensure Int64 dtype and preserve NaT as <NA>
    df_copy['age'] = age.astype(pd.Int64Dtype())
    return df_copy

# def compute_age(df):
#     #yyyy-mm-dd
#     ages = []
#     counter = 0
#     for i in df['date_of_birth']:
#         if i == 'NaN':
#             print('Computer lost')
#             break
#         print(f'{counter}: {i[:4]}')
#         counter += 1


# --- Pipeline Example ---
def prepare_main_player_dataframe() -> pd.DataFrame:
    players = load_players()
    appearances = load_appearances()
    valuations = load_player_valuations()
    stats = aggregate_player_stats(appearances)
    latest = get_latest_valuation(valuations)
    return merge_two(players, merge_two(stats, latest, 'player_id'), 'player_id')


players = load_players()
apps = load_appearances()
events = load_game_events()
gms = load_games()

players = (
    players
    .pipe(add_total_appearances, appearances=apps)
    .pipe(add_total_yellow_cards, appearances=apps)
    .pipe(add_total_red_cards, appearances=apps)
    .pipe(add_total_minutes_played, appearances=apps)
    .pipe(add_total_GA, game_events=events)  # Updated function
    .pipe(add_age)
)
```

```
     data/players.csv
     data/appearances.csv
     data/game_events.csv
     data/games.csv


players['age'] = players['age'] - (2022 - players['last_season'])


# Verify goals are no longer NaN
print(players[players["name"]=="Andrea Pirlo"])



# Columns we want to keep (features only, not target)
keep_cols = ['sub_position', 'position', 'foot',
 'height_in_cm', 'current_club_name', 'total_appearances',
 'total_yellow_cards', 'total_red_cards', 'total_minutes_played',
 'total_goals', 'total_assists', 'age']

# Optional: If you want to retain the target value for modeling later
target_col = 'market_value_in_eur'

# Filter the DataFrame
feature_cols = keep_cols
main_df = players
main_df = main_df[feature_cols + [target_col]].dropna(subset=[target_col]) # Drop r


# Import libraries and utils
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.feature_selection import SelectFromModel
import tensorflow as tf


pd.set_option('display.max_columns', 100)



# ——— 3. Auto-detect numeric vs. categorical among feature_cols ——————————————

# 3a. Numeric columns (int/float/Int64)
num_cols = main_df[feature_cols].select_dtypes(include=['int64','float64','Int64'])

# 3b. Categorical columns (object or category)
cat_cols = main_df[feature_cols].select_dtypes(include=['object','category']).colum

print("Numeric columns that will be scaled:", num_cols)
print("Categorical columns that will be one-hot’ed:", cat_cols)
```

```
    Numeric columns that will be scaled: ['height_in_cm', 'total_appearances', 'tot
    Categorical columns that will be one-hot’ed: ['sub_position', 'position', 'foot
```

```
# ——— 4. Fill/missing + scale numerics, then one-hot encode categoricals ——————————

# Fill missing and scale numeric features
X_num, scaler = fillna_and_scale(main_df, num_cols)

# One-hot encode all categorical features
X_cat, encoder = encode_categorical(main_df, cat_cols)

# Combine into one matrix
import numpy as np

if X_cat.shape[1] > 0:
    X_full = np.hstack([X_num, X_cat.values])
else:
    X_full = X_num

# Target (no NaN, because we dropped them above)
y = main_df['market_value_in_eur']
```

Feature Selecting

```
# Importer XGBoost
from xgboost import XGBRegressor
from sklearn.feature_selection import SelectFromModel

# Opret XGBoost-modellen og fortæl den at bruge GPU'en
# 'gpu_hist' er den moderne og hurtige GPU-algoritme
gbr_gpu = XGBRegressor(tree_method='hist', device='cuda', random_state=42)

# Resten af koden er næsten den samme
gbr_gpu.fit(X_full, y)

# Use 'mean' as the threshold to select features with importance > mean
selector = SelectFromModel(gbr_gpu, prefit=True, threshold='mean')
X_selected = selector.transform(X_full)

print("Total features before selection:", X_full.shape[1])
print("Selected features (importance > mean):", X_selected.shape[1])
```

```
    Total features before selection: 468
    Selected features (importance > mean): 69
```

```
# ——— 6. (Optional) Print exactly which feature names survived ——————————————————
```

```
#     3. (optional) Print exactly which feature names survived

# Build the full list of feature-names in the same order that GBM saw them:
feature_names = num_cols + X_cat.columns.tolist()

# selector.get_support() is a boolean mask of length len(feature_names)

kept_mask          = selector.get_support()
kept_feature_names = [name for name, keep in zip(feature_names, kept_mask) if keep]

print("\nFeatures kept by SelectFromModel (≥ mean importance):")
for feat in kept_feature_names:
    print("  └", feat)
```

lightgbm 1st attemp

```
import lightgbm as lgb
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, ra
```

```
# split off a small val set
X_train_part, X_val, y_train_part, y_val = train_test_split(
    X_train, y_train, test_size=0.2, random_state=42
)

model = lgb.LGBMRegressor(
    n_estimators=2000,
    learning_rate=0.01,
    num_leaves=31,
    reg_alpha=0.1,
    reg_lambda=0.1,
)

# Pass eval_set as before, but move early_stopping_rounds into callbacks
model.fit(
    X_train_part, y_train_part,
    eval_set=[(X_val, y_val)],
    eval_metric='l1',  # or 'rmse', etc.
    callbacks=[ lgb.early_stopping(stopping_rounds=50) ]
)
```

lightgbm 2nd attempt

Double-click (or enter) to edit

```
import lightgbm as lgb
print("Supports GPU?", "gpu" in lgb.LGBMRegressor().get_params())
```

```
     Supports GPU? False
```

Start coding or generate with AI.

```
from sklearn.model_selection import RandomizedSearchCV
import lightgbm as lgb

X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, ra


# Define a parameter grid to sample from
param_dist = {
    'n_estimators': [500, 1000, 2000, 3000],
    'learning_rate': [0.01, 0.05, 0.1],
    'num_leaves': [20, 31, 40, 50],
    'reg_alpha': [0.0, 0.1, 0.5, 1.0],
    'reg_lambda': [0.0, 0.1, 0.5, 1.0],
    'colsample_bytree': [0.7, 0.8, 0.9, 1.0],
    'subsample': [0.7, 0.8, 0.9, 1.0],
    'min_child_samples': [10, 20, 30, 40]
}

# Create a GPU-enabled LightGBM regressor model
lgbm_gpu = lgb.LGBMRegressor(
    random_state=42,
    device_type='gpu',     # use the GPU for training
    gpu_platform_id=0,     # optional: which OpenCL platform to use
    gpu_device_id=0,       # optional: which GPU device to use
    max_bin=63             # recommended for best GPU performance
)

# Set up RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=lgbm_gpu,
    param_distributions=param_dist,
    n_iter=50,                              # number of parameter settings to sample
    cv=3,                                   # 3-fold cross-validation
    scoring='neg_mean_absolute_error',    # evaluation metric
    random_state=42,
    n_jobs=-1,                              # parallelize over CPUs for CV folds
    verbose=2
)

# Fit the random search to the data
random_search.fit(X_train, y_train)

# Print the best parameters and the best score
print("\nBest parameters found:")
print(random_search.best_params_)
print("\nBest MAE found (negated):", -random_search.best_score_)
```

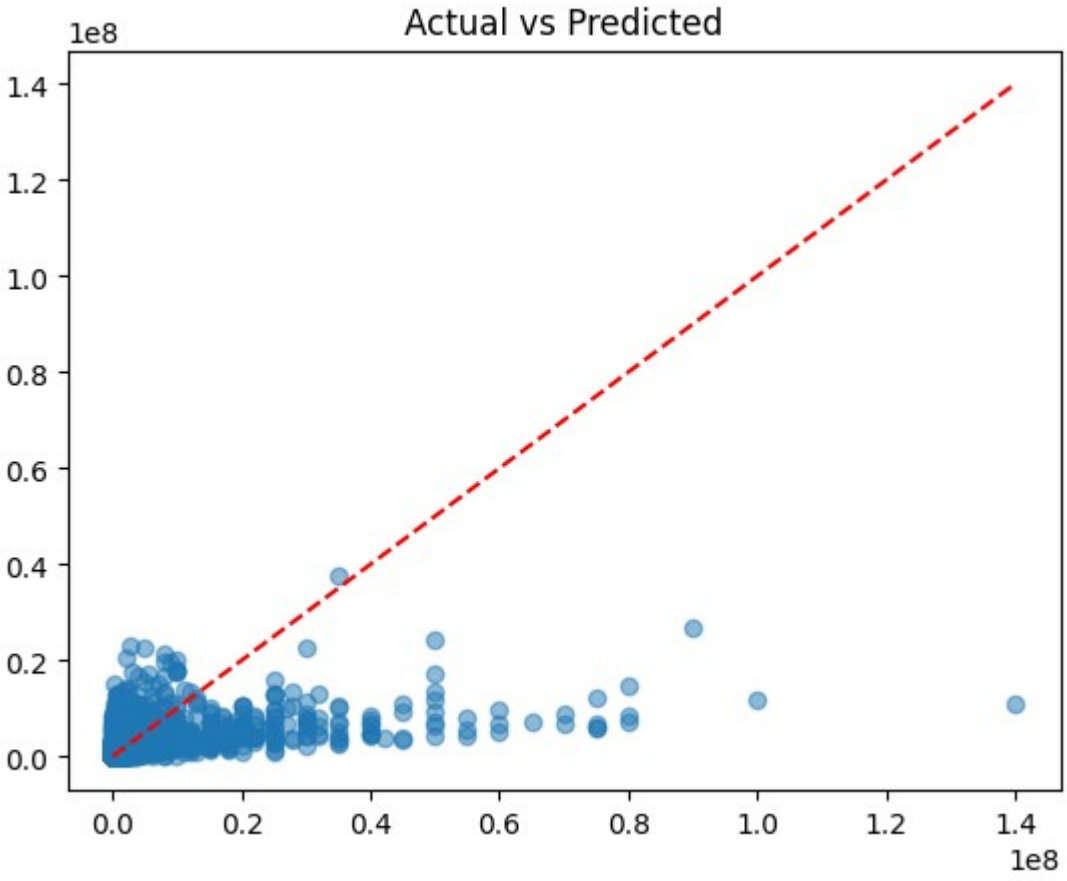
NN 1st attemp

```
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, ra

model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1)
])
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, resto
model.fit(X_train, y_train, validation_split=0.2, epochs=50, batch_size=32, callback

y_pred = model.predict(X_test).ravel()
scatter_actual_vs_pred(y_test, y_pred)
```

NN 2nd attemp

```
from tensorflow.keras import Sequential, layers, callbacks, optimizers

# 1) Build a slightly deeper network with batch-norm and dropout
model = Sequential([
    layers.Input(shape=(X_train.shape[1],)),
    layers.Dense(128, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.3),

    layers.Dense(64, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.2),

    layers.Dense(32, activation='relu'),
    layers.Dense(1)   # regression output
])

# 2) Compile with a sensible initial LR
model.compile(
    optimizer=optimizers.Adam(learning_rate=1e-3),
    loss='mse',
    metrics=['mae']
)

# 3) Callbacks for early-stopping and LR reduction
early_stop = callbacks.EarlyStopping(
    monitor='val_loss',
    patience=15,
    restore_best_weights=True
)
reduce_lr = callbacks.ReduceLROnPlateau(
    monitor='val_loss',
```

```
    factor=0.5,
    patience=5,
    min_lr=1e-6,
    verbose=1
)

# 4) Fit with a higher max-epoch but let early-stop decide
history = model.fit(
    X_train, y_train,
    validation_split=0.2,
    batch_size=64,
    epochs=200,
    callbacks=[early_stop, reduce_lr]
)


scatter_actual_vs_pred(y_test, y_pred)
```




```
print(main_df.shape)
```

```
(31078, 13)
```

```
from sklearn.metrics import mean_absolute_error, r2_score

# Cell 6: Make predictions, clip extremes, compute metrics

# 1. Predict on the test set (log-space)
y_pred_log = model.predict(X_test).flatten()
```

```
# 2. Clip log-predictions to avoid absurd expm1 outputs
max_real_value = 12e7  # €300 million cap
max_log = np.log1p(max_real_value)
y_pred_log_clipped = np.clip(y_pred_log, a_min=0, a_max=max_log)

# 3. Convert back to real-EUR scale
y_test_exp = np.expm1(y_test)
y_pred_exp = np.expm1(y_pred_log_clipped)

# 4. Compute MAE and R^2 on the real-EUR scale
mae_real = mean_absolute_error(y_test_exp, y_pred_exp)
r2_real = r2_score(y_test_exp, y_pred_exp)

# 5. Compute R^2 in log-space
r2_log = r2_score(y_test, y_pred_log)

print(f"Test MAE (EUR scale, clipped): €{mae_real:,.2f}")
print(f"Test R^2 (EUR scale, clipped): {r2_real:.3f}")
print(f"Test R^2 (log1p scale): {r2_log:.3f}")


# Cell 7: Scatter plot of Actual vs. Predicted (real-EUR), filtering out extreme pred

# Only plot points where predicted < €500M, so axes stay readable
limit = 3e8
mask_plot = y_pred_exp < limit

# Apply the same mask to actuals so both arrays align
x_vals = y_test_exp[mask_plot]
y_vals = y_pred_exp[mask_plot]

# Compute min and max for X and Y
x_min, x_max = x_vals.min(), x_vals.max()
y_min, y_max = y_vals.min(), y_vals.max()

# Stretch limits just a tiny bit so points on the border aren't cut off
x_pad = (x_max / x_min) ** 0.05
y_pad = (y_max / y_min) ** 0.05

plt.figure(figsize=(8, 6))
plt.scatter(x_vals, y_vals, alpha=0.3)

# Set log-log scale
plt.xscale('log')
plt.yscale('log')

# Set axis limits to exactly cover the data (with a tiny 1% padding)
plt.xlim(x_min / x_pad, x_max * x_pad)
plt.ylim(y_min / y_pad, y_max * y_pad)

# Draw the y = x reference line over that same range
# On a log-log plot, the diagonal line from (min, min) to (max, max) remains straight
plt.plot(
```

```
    [x_min, x_max],
    [x_min, x_max],
    color='green',
    linewidth=1,
    linestyle='--',
    zorder=0
)

plt.xlabel("Actual Market Value (EUR)")
plt.ylabel("Predicted Market Value (EUR, clipped)")
plt.title("Actual vs. Predicted Market Value (axes fit to data)")

plt.tight_layout()
plt.show()


# Example: try different layer sizes
results = []
for size in [16, 32, 64, 128]:
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(X_train.shape[1],)),
        tf.keras.layers.Dense(size, activation='relu'),
        tf.keras.layers.Dense(size, activation='relu'),
        tf.keras.layers.Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    model.fit(X_train, y_train, epochs=20, batch_size=32, verbose=0)
    y_pred = model.predict(X_test).flatten()
    y_pred_exp = np.expm1(y_pred)
    mae = mean_absolute_error(y_test_exp, y_pred_exp)
    results.append((size, mae))

print("Layer size vs MAE:")
for size, mae in results:
    print(f"  {size} units: MAE = €{mae:,.2f}")



from sklearn.metrics import mean_absolute_error, r2_score

# Make predictions on the test set
y_pred_lgbm = model.predict(X_test)

# Calculate evaluation metrics
mae_lgbm = mean_absolute_error(y_test, y_pred_lgbm)
r2_lgbm = r2_score(y_test, y_pred_lgbm)

print(f"LightGBM Test MAE: {mae_lgbm:,.2f}")
print(f"LightGBM Test R^2: {r2_lgbm:.3f}")

# Visualize actual vs. predicted
scatter_actual_vs_pred(y_test, y_pred_lgbm, title='LightGBM: Actual vs Predicted Ma


from sklearn.model_selection import RandomizedSearchCV
```

```
import lightgbm as lgb

# Define a parameter grid to sample from
param_dist = {
    'n_estimators': [500, 1000, 2000, 3000],
    'learning_rate': [0.01, 0.05, 0.1],
    'num_leaves': [20, 31, 40, 50],
    'reg_alpha': [0.0, 0.1, 0.5, 1.0],
    'reg_lambda': [0.0, 0.1, 0.5, 1.0],
    'colsample_bytree': [0.7, 0.8, 0.9, 1.0],
    'subsample': [0.7, 0.8, 0.9, 1.0],
    'min_child_samples': [10, 20, 30, 40]
}

# Create a GPU-enabled LightGBM regressor model
lgbm_gpu = lgb.LGBMRegressor(
    random_state=42,
    device_type='gpu',     # use the GPU for training
    gpu_platform_id=0,     # optional: which OpenCL platform to use
    gpu_device_id=0,       # optional: which GPU device to use
    max_bin=63             # recommended for best GPU performance
)

# Set up RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=lgbm_gpu,
    param_distributions=param_dist,
    n_iter=50,                              # number of parameter settings to sample
    cv=3,                                   # 3-fold cross-validation
    scoring='neg_mean_absolute_error',    # evaluation metric
    random_state=42,
    n_jobs=-1,                              # parallelize over CPUs for CV folds
    verbose=2
)

# Fit the random search to the data
random_search.fit(X_train, y_train)

# Print the best parameters and the best score
print("\nBest parameters found:")
print(random_search.best_params_)
print("\nBest MAE found (negated):", -random_search.best_score_)
```