PREDITING THE SUCCESS OF FINANCIAL AND ACCOUNTING COURSES ON UDEMY

A **Data Analyst** project by

AMIT BHADE

Tools Used – ***Google Colab, Excel***

Techniques – ***Python, SQL, Machine Learning***

Objectives

Problem Statement

Predict the number of subscribers for Finance & Accounting courses based on course features (ratings, reviews, lectures, price, etc.) and analyze factors that drive course popularity.

Dataset

Click here to download the Dataset

ABOUT DATASET:

- The Dataset contains information such as course ID, title, URL, subscription details, pricing, and course content metrics.
- Columns: id, title, url, is_paid, num_subscribers, avg_rating, avg_rating_recent, rating, num_reviews, is_wishlisted, num_published_lectures, num_published_practice_tests, created, published_time, discount_price__amount, discount_price__currency, discount_price__price_string,price_detail__amount, price_detail__currency, price_detail__price_string

Project Summery

The goal of this project was to predict the number of subscribers for Finance & Accounting courses on Udemy based on course characteristics such as ratings, reviews, price, and course features. This task demonstrates how predictive analytics can be applied in real-world e-learning datasets.

Approach

1. **Data Loading & Preprocessing**
    - Imported the dataset and handled missing values.
    - Normalized column names, converted date fields, and engineered features (e.g., course age, reviews per subscriber, discount %).
    - Converted categorical variables (e.g., is_paid) using one-hot encoding.

2. **Feature Selection**
   - Selected meaningful predictors: ratings, recent ratings, reviews, price, lectures, practice tests, course age, discounts, and paid/free status.
   - Focused on features most likely to influence subscriber counts.
3. **Model Training**
   - Trained a **Random Forest Regressor**, a flexible, non-linear algorithm well-suited to regression problems.
   - Compared results with a Ridge Regression baseline.
4. **Model Evaluation**
   - Evaluated using **R^2 (explained variance)** and **MAE/MSE (error metrics)**.
   - Random Forest performed better than Ridge, capturing non-linear relationships between features and subscriber counts.
5. **Visualization**
   - Growth of Finance & Accounting courses over time.
   - Distributions of ratings, reviews, and subscribers.
   - Scatter plots showing relationships (e.g., reviews vs subscribers).
   - Visualization of actual vs predicted subscribers for evaluation.

Step By Step Project Implementation

Step 1: Import Libraries

```
import pandas as pd
import numpy as np
import re
from datetime import datetime

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_absolute_error
```

Step 2: Load Dataset

```
from google.colab import files
uploaded = files.upload()

print("Shape:", df_raw.shape)
df_raw.info()
df_raw.sample(5)
```

Step 3: Data Preprocessing

```
df = df_raw.copy()

# Normalize column names
def to_snake(s):
    s = re.sub(r'[^0-9a-zA-Z]+', '_', s).strip('_')
    return s.lower()
df.columns = [to_snake(c) for c in df.columns]

# Common aliases → standard names
rename_map = {
    'published_time': 'published_at',
    'published_time_utc': 'published_at',
    'published_date': 'published_at',
    'created': 'created_at',
    'creation_time': 'created_at',
    'num_published_lectures': 'num_lectures',
    'num_published_practice_tests': 'num_practice_tests',
    'price_detail_amount': 'price_amount',
    'price_detail_currency': 'price_currency',
    'discounted_price_amount': 'discount_amount',
    'discounted_price_currency': 'discount_currency',
    'rating': 'avg_rating',
    'recent_rating': 'avg_rating_recent',
    'reviews': 'num_reviews',
    'subscribers': 'num_subscribers'
}
for k, v in rename_map.items():
    if k in df.columns and v not in df.columns:
        df.rename(columns={k: v}, inplace=True)

# Parse dates
for dc in ['created_at', 'published_at']:
    if dc in df.columns:
        df[dc] = pd.to_datetime(df[dc], errors='coerce')

# Ensure is_paid exists or infer from price
```

```
if 'is_paid' in df.columns:
    df['is_paid'] = (
        df['is_paid']
        .astype(str).str.lower()
        .map({'true': True, 'false': False, '1': True, '0': False})
        .fillna(df['is_paid'])
    )
else:
    df['is_paid'] = np.where(df.get('price_amount', 0).fillna(0) > 0, True, False)

# Parse price from string if needed (e.g., "₹8,640")
def parse_price_string(s):
    if pd.isna(s):
        return np.nan, None
    currency_match = re.findall(r'[^\d\s.,]+', str(s))
    currency_code = currency_match[0] if currency_match else None
    number = re.sub(r'[^\d.,]', '', str(s))
    digits_only = re.sub(r'[^\d]', '', number)
    return (float(digits_only) if digits_only else np.nan, currency_code)

if 'price_amount' not in df.columns and 'price_detail_price_string' in df.columns:
    parsed = df['price_detail_price_string'].apply(parse_price_string)
    df['price_amount'] = parsed.apply(lambda x: x[0])
    df['price_currency'] = parsed.apply(lambda x: x[1])

# Drop duplicates
before = df.shape[0]
key_cols = [c for c in ['id', 'url', 'title'] if c in df.columns]
if 'id' in df.columns:
    df = df.drop_duplicates(subset=['id'])
elif len(key_cols) >= 2:
    df = df.drop_duplicates(subset=key_cols)
else:
    df = df.drop_duplicates()
print(f"Dropped duplicates: {before - df.shape[0]}")

df.head(3)
```

Step 4: Feature engineering

```
# Year & course age (use published_at if available, else created_at)
if 'published_at' in df.columns or 'created_at' in df.columns:
    pub = df['published_at'] if 'published_at' in df.columns else pd.NaT
    cre = df['created_at'] if 'created_at' in df.columns else pd.NaT
    chosen = np.where(pd.notna(pub), pub, cre)
    dt = pd.to_datetime(chosen, errors='coerce')
    df['year'] = dt.dt.year
    base = pd.to_datetime('today')
```

```
    df['course_age_years'] = (base - dt).dt.days / 365.25
else:
    df['year'] = np.nan
    df['course_age_years'] = np.nan

# Reviews per subscriber
if 'num_reviews' in df.columns and 'num_subscribers' in df.columns:
    denom = df['num_subscribers'].replace({0: np.nan})
    df['reviews_per_sub'] = df['num_reviews'] / denom

# Recent vs overall rating
if 'avg_rating_recent' in df.columns and 'avg_rating' in df.columns:
    df['rating_recent_diff'] = df['avg_rating_recent'] - df['avg_rating']

# Lectures per practice test
if 'num_lectures' in df.columns and 'num_practice_tests' in df.columns:
    denom = df['num_practice_tests'].replace({0: np.nan})
    df['lectures_per_test'] = df['num_lectures'] / denom

# Discount percentage
if 'discount_amount' in df.columns and 'price_amount' in df.columns:
    with np.errstate(divide='ignore', invalid='ignore'):
        df['discount_percentage'] = (df['discount_amount'] / df['price_amount']) * 100
else:
    # If we only know free vs paid, mark free as 100% discount; paid unknown
    df['discount_percentage'] = np.where(df['is_paid'] == False, 100.0, np.nan)

df.head(3)
```

Step 5: EDA Growth over time

```
growth = df.dropna(subset=['year']).groupby('year').size().reset_index(name='courses')
display(growth.head())

plt.figure(figsize=(8,4))
plt.plot(growth['year'], growth['courses'])
plt.title('Finance & Accounting Courses: Growth Over Time')
plt.xlabel('Year')
plt.ylabel('Number of Courses')
plt.tight_layout()
plt.show()
```

Step 6: EDA Course characteristics

```
# Helper to plot a 1D histogram
def plot_hist(series, title, xlabel):
    s = series.dropna()
    if s.empty:
        print(f"Skipping {title}: no data.")
```

```
        return
    plt.figure(figsize=(7,4))
    plt.hist(s, bins=30)
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel('Count')
    plt.tight_layout()
    plt.show()

# Ratings
if 'avg_rating' in df.columns:
    plot_hist(df['avg_rating'], 'Average Rating Distribution', 'avg_rating')
if 'avg_rating_recent' in df.columns:
    plot_hist(df['avg_rating_recent'], 'Recent Rating Distribution', 'avg_rating_recent')

# Reviews, Subscribers (use log scale quick view)
if 'num_reviews' in df.columns:
    plot_hist(np.log1p(df['num_reviews']), 'Log(1+Reviews) Distribution',
'log1p(num_reviews)')
if 'num_subscribers' in df.columns:
    plot_hist(np.log1p(df['num_subscribers']), 'Log(1+Subscribers) Distribution',
'log1p(num_subscribers)')

# Lectures & practice tests
if 'num_lectures' in df.columns:
    plot_hist(df['num_lectures'], 'Number of Lectures Distribution', 'num_lectures')
if 'num_practice_tests' in df.columns:
    plot_hist(df['num_practice_tests'], 'Practice Tests Distribution', 'num_practice_tests')

# Reviews per subscriber
if 'reviews_per_sub' in df.columns:
    plot_hist(df['reviews_per_sub'], 'Reviews per Subscriber', 'reviews_per_sub')
```

Step 7: EDA — Relationships

```
def scatter(x, y, title, xlabel, ylabel):
    x_s = df[x]
    y_s = df[y]
    mask = x_s.notna() & y_s.notna()
    if mask.sum() == 0:
        print(f"Skipping {title}: no overlapping data.")
        return
    plt.figure(figsize=(7,5))
    plt.scatter(x_s[mask], y_s[mask], alpha=0.3, s=10)
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.tight_layout()
    plt.show()
```

```
if set(['num_reviews','num_subscribers']).issubset(df.columns):
    scatter('num_reviews','num_subscribers','Reviews vs 
Subscribers','num_reviews','num_subscribers')

if set(['avg_rating','num_subscribers']).issubset(df.columns):
    scatter('avg_rating','num_subscribers','Rating vs 
Subscribers','avg_rating','num_subscribers')

if set(['price_amount','num_subscribers']).issubset(df.columns):
    scatter('price_amount','num_subscribers','Price vs 
Subscribers','price_amount','num_subscribers')
```

Step 9: EDA Paid vs Free + Discount overview

```
# Paid vs Free counts
paid_counts = df['is_paid'].value_counts(dropna=False)
display(paid_counts)

plt.figure(figsize=(6,4))
plt.bar(['Free','Paid'], [paid_counts.get(False,0), paid_counts.get(True,0)])
plt.title('Paid vs Free (Finance & Accounting)')
plt.xlabel('Type')
plt.ylabel('Count of Courses')
plt.tight_layout()
plt.show()

# Discount percentage distribution
if 'discount_percentage' in df.columns:
    plot_hist(df['discount_percentage'], 'Discount Percentage Distribution', 
'discount_percentage (%)')
```

Step 9: Correlations

```
num_df = df.select_dtypes(include=[np.number]).copy()
if not num_df.empty:
    corr = num_df.corr(numeric_only=True)
    display(corr[['num_subscribers']].sort_values('num_subscribers', 
ascending=False).head(15))
else:
    print("No numeric columns found for correlation.")
```

Step 10: Modeling Predict num_subscribers

```
target = 'num_subscribers'
base_features = [
    'avg_rating', 'avg_rating_recent', 'num_reviews',
    'price_amount', 'course_age_years',
    'num_lectures', 'num_practice_tests',
    'discount_percentage', 'is_paid'
```

```
]

# Keep only existing features
features = [f for f in base_features if f in df.columns]
print("Using features:", features)

# Filter data with target present
data = df.dropna(subset=[target])[[target] + features].copy()

# Remove negative/invalid targets (safety)
data = data[data[target] >= 0]

X = data[features]
y = data[target]

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=RANDOM_STATE
)

# Identify columns by type
num_cols = [c for c in X.columns if pd.api.types.is_numeric_dtype(X[c])]
cat_cols = [c for c in X.columns if not pd.api.types.is_numeric_dtype(X[c])]

preprocess = ColumnTransformer(
    transformers=[
        ('num', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='median')),
            ('scaler', StandardScaler())
        ]), num_cols),
        ('cat', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='most_frequent')),
            ('onehot', OneHotEncoder(handle_unknown='ignore'))
        ]), cat_cols)
    ],
    remainder='drop'
)

# Model 1: Ridge
ridge = Pipeline(steps=[('prep', preprocess), ('model', Ridge())])
ridge.fit(X_train, y_train)
pred_r = ridge.predict(X_test)
print("Ridge -> R2:", r2_score(y_test, pred_r), " MAE:", mean_absolute_error(y_test,
pred_r))

# Model 2: Random Forest
rf = Pipeline(steps=[('prep', preprocess),
                     ('model', RandomForestRegressor(
                         n_estimators=500, random_state=RANDOM_STATE, n_jobs=-1))]
)
```

```
rf.fit(X_train, y_train)
pred_rf = rf.predict(X_test)
print("RandomForest -> R2:", r2_score(y_test, pred_rf), " MAE:",
mean_absolute_error(y_test, pred_rf))
```

Step 11: Feature importance (Random Forest)

```
# Extract post-encoding feature names
num_feats = num_cols
cat_feat_names = []
if cat_cols:
    ohe = rf.named_steps['prep'].named_transformers_['cat'].named_steps['onehot']
    cat_feat_names = ohe.get_feature_names_out(cat_cols).tolist()

all_feat_names = num_feats + cat_feat_names

rf_model = rf.named_steps['model']
if hasattr(rf_model, "feature_importances_"):
    importances = rf_model.feature_importances_
    imp_df = pd.DataFrame({'feature': all_feat_names, 'importance':
importances}).sort_values('importance', ascending=False)
    display(imp_df.head(20))

    plt.figure(figsize=(8,5))
    topn = imp_df.head(15)
    plt.barh(topn['feature'][::-1], topn['importance'][::-1])
    plt.title('Random Forest Feature Importance (Top 15)')
    plt.xlabel('Importance')
    plt.ylabel('Feature')
    plt.tight_layout()
    plt.show()
else:
    print("Feature importances not available for this model.")
```

Step 12: Modeling — Predict avg_rating instead

```
target2 = 'avg_rating'
if target2 in df.columns:
    base_features2 = [
        'avg_rating_recent', 'num_reviews', 'price_amount',
        'course_age_years', 'num_lectures', 'num_practice_tests',
        'discount_percentage', 'is_paid'
    ]
    features2 = [f for f in base_features2 if f in df.columns]
    print("Using features for rating:", features2)

    data2 = df.dropna(subset=[target2])[[target2] + features2].copy()
    X2 = data2[features2]
    y2 = data2[target2]
```

```
    num_cols2 = [c for c in X2.columns if pd.api.types.is_numeric_dtype(X2[c])]
    cat_cols2 = [c for c in X2.columns if not pd.api.types.is_numeric_dtype(X2[c])]

    preprocess2 = ColumnTransformer(
        transformers=[
            ('num', Pipeline(steps=[
                ('imputer', SimpleImputer(strategy='median')),
                ('scaler', StandardScaler())
            ]), num_cols2),
            ('cat', Pipeline(steps=[
                ('imputer', SimpleImputer(strategy='most_frequent')),
                ('onehot', OneHotEncoder(handle_unknown='ignore'))
            ]), cat_cols2)
        ],
        remainder='drop'
    )

    X2_train, X2_test, y2_train, y2_test = train_test_split(
        X2, y2, test_size=0.2, random_state=RANDOM_STATE
    )

    ridge2 = Pipeline(steps=[('prep', preprocess2), ('model', Ridge())])
    ridge2.fit(X2_train, y2_train)
    pred_r2 = ridge2.predict(X2_test)
    print("Ridge (rating) -> R2:", r2_score(y2_test, pred_r2), " MAE:",
mean_absolute_error(y2_test, pred_r2))

    rf2 = Pipeline(steps=[('prep', preprocess2),
                    ('model', RandomForestRegressor(
                        n_estimators=400, random_state=RANDOM_STATE, n_jobs=-1))]
    )
    rf2.fit(X2_train, y2_train)
    pred_rf2 = rf2.predict(X2_test)
    print("RandomForest (rating) -> R2:", r2_score(y2_test, pred_rf2), " MAE:",
mean_absolute_error(y2_test, pred_rf2))
else:
    print("avg_rating not found — skipping this optional model.")
```

Step 13: Save cleaned dataset

```
df.to_csv('Cleaned_Finance_Accounting_Udemy.csv', index=False)
print("Saved: Cleaned_Finance_Accounting_Udemy.csv")
```

Key Insights

- Course popularity is skewed — a few flagship courses capture the majority of subscribers.
- Reviews and ratings are strong predictors of subscriber numbers.
- Price and discounts play a relatively smaller role compared to quality signals.
- Predictive models explain some variance, but external factors (marketing, instructor reputation, Udemy promotion) also heavily influence success.

Conclusion

This project illustrates how regression techniques can be applied to predict course success in online learning. Random Forest provided a good balance of interpretability and predictive power.

Reference