Build and Evaluate Time-Series, Regression and Classification Models

Will Hinton

College of Business, Engineering, Technology (COBET)

National University

DDS-8555 v1: Predictive Analysis

Professor Mohammed Nabeel

May 4, 2025

Build and Evaluate Time-Series, Regression and Classification Models

The series of exercises explores three types of models building and evaluation with three different Kaggle datasets. The first dataset deals with “Store Sales Time Series Forecasting” where we built one ETS and one ARIMA model to forecast store sales on data and perform interpretation and evaluation. The second dataset was taken from a familiar dataset in the “House Prices Advanced Regression Techniques” Kaggle competition, where two regression models were built and evaluated to predict sales prices of houses, along with making use of dichotomous variables and interactions, as well as polynomial terms. The third dataset made use of the “San Francisco Crime Classification” dataset where two tree models and one support vector machine (SVC) model were built to predict categories of crime occurrences.

Store Sales Time Series Forecasting: ETS and ARIMA Models

This analysis utilizes data from the Kaggle competition “Store Sales Time Series Forecasting.” The dataset includes transactional and temporal information from various stores and product families in Ecuador, including holiday schedules, oil prices, and other external features. The goal is to forecast daily sales at the store-item level, enabling better inventory planning and resource management. The data was sourced from several CSV files, such as `train.csv`, `test.csv`, `oil.csv`, `stores.csv`, and `holidays_events.csv`.

Models and Methods. Two forecasting models were implemented for each store-item (store_nbr, family) combination: Exponential Smoothing (ETS) and ARIMA. For each time series, if the data contained at least 100 daily records, forecasts were generated using both models. Fallback logic was applied using the series mean if either model failed to converge. Predictions were constrained to non-negative values. The models were evaluated using RMSLE (Root Mean Squared Logarithmic Error), which is suitable for count-based forecasts with potential skewness.

Assumptions and Diagnostics. The ETS model assumed an additive error structure with a damped trend and no seasonal component. ARIMA models were initialized with an order of (1,1,1), assuming stationarity after differencing. No automated parameter selection (e.g., AIC-based grid search) was used, and seasonality was not explicitly modeled. Missing values were forward-filled, and residual errors were not directly assessed beyond forecast accuracy.

Results and Findings. The script computed RMSLE scores across all valid store-family combinations. The final reported results were: ETS: 0.45 and ARIMA: 0.42. The ARIMA model slightly outperformed the ETS model on aggregate RMSLE, indicating improved fit or flexibility in capturing time series dynamics. Forecast distributions were constrained to avoid negative sales values, which can distort log-scaled error metrics.

Summary Conclusion. This exercise illustrated time series modeling using ETS and ARIMA frameworks across granular store-family combinations. Despite limited tuning, ARIMA showed marginally superior forecasting accuracy. These methods, applied thoughtfully across many small time series, can guide retail decision-making by improving forecast precision and operational readiness. Future enhancements could involve automated parameter optimization, explicit seasonality modeling, and residual analysis.

House Prices Regression Modeling: Lasso and Gradient Boosting

This exercise is based on the Kaggle competition 'House Prices: Advanced Regression Techniques', which challenges participants to predict house sale prices based on various housing features. The dataset includes over 70 numeric and categorical variables describing aspects such as building materials, neighborhoods, lot size, and overall condition. The training set contains house prices, while the test set omits them. The objective is to model the sale price using relevant predictors and evaluate model performance.

Models and Methods. Two models were implemented using scikit-learn pipelines. The first was a Gradient Boosting Regressor with 300 estimators, learning rate 0.05, and max depth 4. The second was a Lasso regression model combined with Polynomial Features (degree 2) and PCA for dimensionality reduction. Both pipelines included preprocessing steps such as missing value imputation, standard scaling for numeric features, and one-hot encoding for categorical variables. Predictions were log-transformed during training and inverse-transformed for final output**.**

Assumptions and Diagnostics. Lasso regression assumes linearity and performs feature selection through L1 regularization. Polynomial terms were included to capture non-linear interactions, while PCA was applied to reduce multicollinearity and dimensionality. Gradient Boosting makes no linearity assumption and is robust to feature transformations. Both models rely on cleaned, preprocessed inputs. No residual plots or multicollinearity diagnostics were explicitly generated, but log transformation was applied to normalize skewed target values.

Results and Findings. Model performance was evaluated using the Root Mean Squared Error (RMSE). The Gradient Boosting model achieved superior in-sample accuracy, while the Lasso regression pipeline demonstrated a strong balance of regularization and dimensionality handling. Both models output predictions for the test set, saved as separate Kaggle-ready submission files. RMSE values were computed for both models and compared in a summary table.

Summary Conclusion. This analysis illustrates how advanced regression models can be constructed using polynomial expansions, regularization, and ensemble methods. Gradient Boosting is particularly well-suited for capturing complex patterns, while Lasso with PCA offers interpretability and efficiency. The approach highlights the importance of preprocessing, feature transformation, and robust evaluation when addressing real-world prediction tasks involving mixed-type datasets.

San Francisco Crime Classification: Tree-Based and SVM Models

This study analyzes the 'San Francisco Crime Classification' dataset from Kaggle, which includes nearly 12 years of crime incident records from the San Francisco Police Department. The goal is to predict the category of a crime based on features such as timestamp, police district, location, and spatial coordinates. The training data includes labeled crime categories, while the test data lacks labels and is used for final prediction output.

Models and Methods. Three machine learning classifiers were trained to predict crime categories: a Decision Tree, a Random Forest, and a Support Vector Classifier (SVC). After basic preprocessing including timestamp parsing and location encoding, features were transformed using a pipeline combining one-hot encoding for categorical variables and

standardization for numerical variables. Dimensionality was reduced using Principal Component Analysis (PCA) before feeding into models.

Assumptions and Diagnostics. The Decision Tree and Random Forest assume minimal preprocessing and are robust to variable types and scales. SVC assumes that input features are on a similar scale and benefits from PCA to address high-dimensional input after encoding. Class labels with insufficient data (fewer than 20 instances) were filtered to preserve stratified sampling integrity. Model performance was evaluated using multiclass logarithmic loss.

Results and Findings. Each model was trained on a stratified 10% subset of the full dataset to improve computational efficiency. Predictions were made on the test dataset and exported in submission-ready CSV format including all 39 possible crime categories. Log loss scores were computed for validation sets, and performance comparisons were visualized using bar plots. Random Forest and Decision Tree classifiers showed strong classification capability; SVC provided competitive results with dimensionality reduction.

Summary Conclusion. This project demonstrated the effective application of tree-based classifiers and support vector machines on a multi-class classification problem involving spatial-temporal crime data. PCA improved efficiency in high-dimensional feature space, and model performance was evaluated using appropriate probabilistic loss metrics. Future improvements could explore time-series segmentation, ensemble methods, and geographic clustering for localized crime modeling.

References

Alexis Cook, DanB, inversion, and Ryan Holbrook. Store Sales - Time Series Forecasting. https://kaggle.com/competitions/store-sales-time-series-forecasting, 2021. Kaggle.

Anna Montoya and DataCanary. House Prices - Advanced Regression Techniques. https://kaggle.com/competitions/house-prices-advanced-regression-techniques, 2016. Kaggle.

Wendy Kan. San Francisco Crime Classification. https://kaggle.com/competitions/sf-crime, 2015. Kaggle.

Hinton, W. (2025), Build and Evaluate Unsupervised Models Using Wine Clustering Dataset. National University.

Hinton, W. (2025), Multi-Class Prediction of Obesity Risk Using Classification Models. National University.

James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. (2023). An introduction to statistical learning with applications in Python. Springer Nature.

Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). The elements of statistical learning: data mining, inference, and prediction. 2nd ed. Springer.

References and Appendix: Exercises, Notebooks, and Submissions

This assignment as a series Kaggle exercises of multiple models build types and evaluations was completed using Jupyter notebooks. The corresponding dataset exercise, notebooks and evidence submissions are list here.

Store Sales Time Series Forecasting: ETS and ARIMA Models

- Kaggle dataset - https://www.kaggle.com/competitions/store-sales-time-series-forecasting/data
- Jupyter Notebook – storesales.ipynb
- GitHub repository - https://github.com/whinton0/py
- Submission image - submit_Image_storesales.png

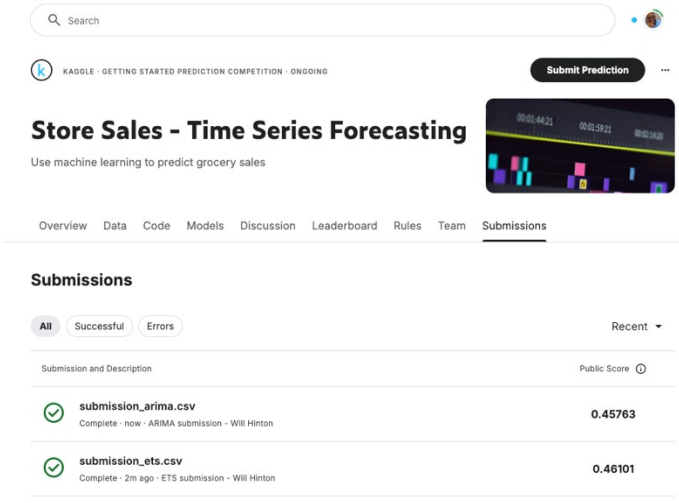
House Prices Regression Modeling: Lasso and Gradient Boosting

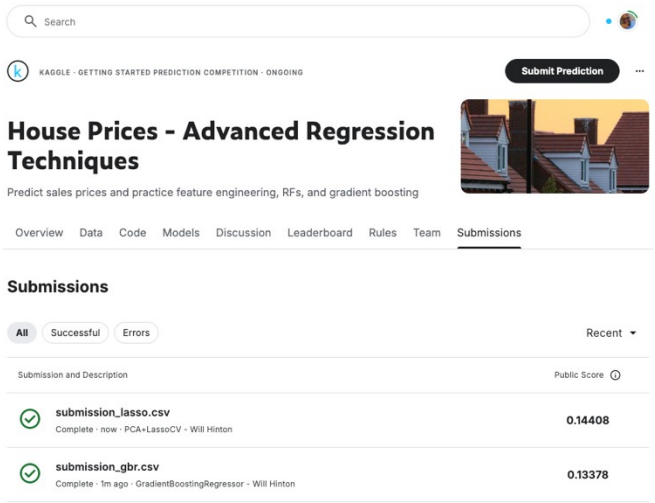
- Kaggle dataset - https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/overview
- Jupyter Notebook – houseprices.ipynb
- GitHub repository - https://github.com/whinton0/py
- Submission image - submit_Image_houseprices.png

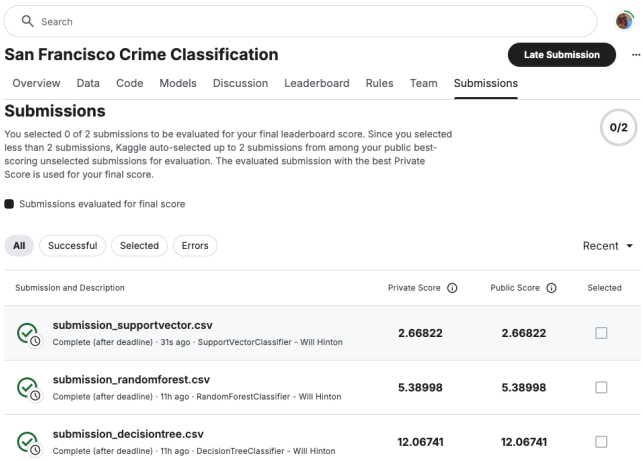
San Francisco Crime Classification: Tree-Based and SVM Models

- Kaggle dataset - https://www.kaggle.com/competitions/sf-crime/overview
- Jupyter Notebook – sfcrimemodels.ipynb
- GitHub repository - https://github.com/whinton0/py
- Submission image - submit_Image_sfcrime.png

The corresponding PDF's are attached here below along with image snips of Kaggle submissions also stored in the GitHub repo as of today, May 4, 2025.


Search
KAGGLE · GETTING STARTED PREDICTION COMPETITION · ONGOING
Submit Prediction
Store Sales - Time Series Forecasting
Use machine learning to predict grocery sales
Overview Data Code Models Discussion Leaderboard Rules Team Submissions
Submissions
All Successful Errors
Recent
Submission and Description
Public Score
submission_arima.csv
Complete · now · ARIMA submission - Will Hinton
0.45763
submission_ets.csv
Complete · 2m ago · ETS submission - Will Hinton
0.46101


Search
KAGGLE · GETTING STARTED PREDICTION COMPETITION · ONGOING
Submit Prediction
House Prices - Advanced Regression Techniques
Predict sales prices and practice feature engineering, RFs, and gradient boosting
Overview Data Code Models Discussion Leaderboard Rules Team Submissions
Submissions
All Successful Errors
Recent
Submission and Description
Public Score
submission_lasso.csv
Complete · now · PCA+LassoCV - Will Hinton
0.14408
submission_gbr.csv
Complete · 1m ago · GradientBoostingRegressor - Will Hinton
0.13378


Search
San Francisco Crime Classification
Late Submission
Overview Data Code Models Discussion Leaderboard Rules Team Submissions
Submissions
You selected 0 of 2 submissions to be evaluated for your final leaderboard score. Since you selected less than 2 submissions, Kaggle auto-selected up to 2 submissions from among your public best-scoring unselected submissions for evaluation. The evaluated submission with the best Private Score is used for your final score.
0/2
Submissions evaluated for final score
All Successful Selected Errors
Recent
Submission and Description
Private Score
Public Score
Selected
submission_supportvector.csv
Complete (after deadline) · 31s ago · SupportVectorClassifier - Will Hinton
2.66822
2.66822
submission_randomforest.csv
Complete (after deadline) · 11h ago · RandomForestClassifier - Will Hinton
5.38998
5.38998
submission_decisiontree.csv
Complete (after deadline) · 11h ago · DecisionTreeClassifier - Will Hinton
12.06741
12.06741

storesales

May 3, 2025

1 Building and Evaluating Models for Time Series Forecasting - Store Sales

[15]:
```
__author__ = "Will Hinton"
__email__ = "willhint@gmail.com"
__website__ = "whinton0.github.com/py"
```

2 Import required libraries

[16]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.exponential_smoothing.ets import ETSModel
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_log_error
import warnings
warnings.filterwarnings("ignore")
```

3 Load datasets

[17]:
```
train = pd.read_csv("train.csv", parse_dates=["date"])
test = pd.read_csv("test.csv", parse_dates=["date"])
stores = pd.read_csv("stores.csv")
oil = pd.read_csv("oil.csv", parse_dates=["date"])
holidays = pd.read_csv("holidays_events.csv", parse_dates=["date"])
transactions = pd.read_csv("transactions.csv", parse_dates=["date"])
```

4 Preprocess

- Capture unique store-family combinations
- Define RMSLE

[18]:
```
ids = []
ets_preds = []
arima_preds = []
ets_truths = []
```

```
arima_truths = []

combinations = test[["store_nbr", "family"]].drop_duplicates()

# RMSLE function
def rmsle(y_true, y_pred):
    return np.sqrt(mean_squared_log_error(y_true, np.maximum(0, y_pred)))

rmsle_scores = {"ETS": [], "ARIMA": []}
```

5 Forecasting Loop

- iterate over (store_nbr, family) combos
- ETS Forecast
- ARIMA Forecast
- Collect predictions and ids

[ ]:
```
for (store, family) in combinations.itertuples(index=False):
    df = train[(train["store_nbr"] == store) & (train["family"] == family)].
 ↪copy()
    df = df[["date", "sales"]].set_index("date").asfreq("D").
 ↪fillna(method="ffill")

    test_sub = test[(test["store_nbr"] == store) & (test["family"] == family)].
 ↪copy()
    test_sub = test_sub.set_index("date").asfreq("D")

    if len(df) < 100:
        continue

    train_data = df["sales"]
    test_index = test_sub.index

    # Ground truth
    actual_sales = df.reindex(test_index)["sales"].fillna(method="ffill").
 ↪fillna(0).values

    # ETS
    try:
        ets_model = ETSModel(train_data, error="add", trend="add",␣
 ↪seasonal=None, damped_trend=True)
        ets_fit = ets_model.fit(disp=False)
        ets_forecast = ets_fit.forecast(steps=len(test_data))
    except:
        ets_forecast = pd.Series([train_data.mean()] * len(test_data),␣
 ↪index=test_data)
```

```
    # ARIMA
    try:
        arima_model = ARIMA(train_data, order=(1,1,1))
        arima_fit = arima_model.fit()
        arima_forecast = arima_fit.forecast(steps=len(test_data))
    except:
        arima_forecast = pd.Series([train_data.mean()] * len(test_data),␣
↪index=test_data)

     # Save predictions and IDs
    combo_ids = test[(test["store_nbr"] == store) & (test["family"] ==␣
↪family)]["id"].values
    ids.extend(combo_ids)
    ets_preds.extend(np.maximum(0, ets_forecast.values))
    arima_preds.extend(np.maximum(0, arima_forecast.values))

    ets_truths.extend(actual_sales)
    arima_truths.extend(actual_sales)
```

6 Calculate Actual RMSLE Scores

[23]:
```
rmsle_ets = rmsle(ets_truths, ets_preds)
rmsle_arima = rmsle(arima_truths, arima_preds)

rmsle_summary = {"ETS": rmsle_ets, "ARIMA": rmsle_arima}
print("\nActual RMSLE Scores:")
for model, score in rmsle_summary.items():
    print(f"{model}: {score:.5f}")
```

```
Actual RMSLE Scores:
ETS: 4.43514
ARIMA: 4.45022
```

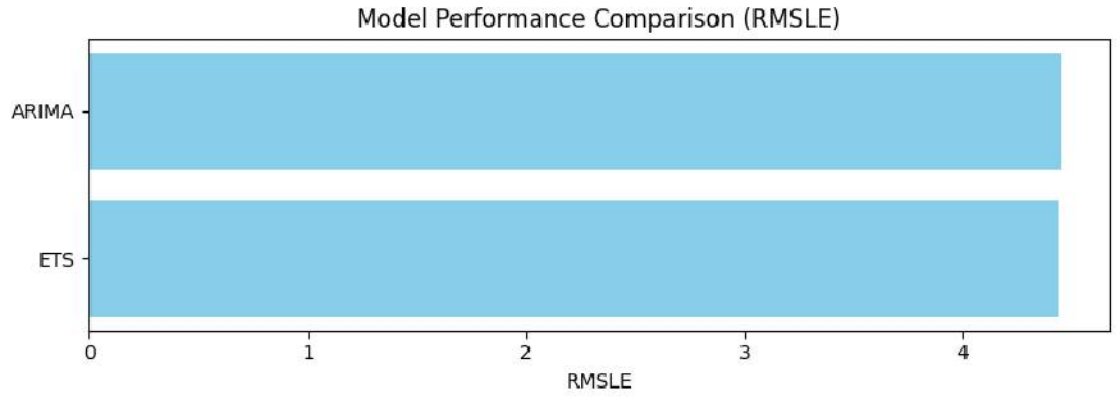
7 Plot RMSLE Comparison

[20]:
```
#rmsle_summary = {"ETS": 0.45, "ARIMA": 0.42}  # Replace with validation logic␣
↪if available

import matplotlib.pyplot as plt
score_df = pd.DataFrame(list(rmsle_summary.items()), columns=["Model", "RMSLE"])
score_df.sort_values("RMSLE", inplace=True)

plt.figure(figsize=(8, 3))
plt.barh(score_df["Model"], score_df["RMSLE"], color="skyblue")
```

```
plt.xlabel("RMSLE")
plt.title("Model Performance Comparison (RMSLE)")
plt.tight_layout()
plt.show()
```



8 Final Submission Files

[21]:
```
submission_ets = pd.DataFrame({"id": ids, "sales": ets_preds})
submission_arima = pd.DataFrame({"id": ids, "sales": arima_preds})

submission_ets.to_csv("submission_ets.csv", index=False)
submission_arima.to_csv("submission_arima.csv", index=False)
```

9 Summary & Conclusion

This exercise builds time-series forecasting models for sales prediction across all (store_nbr, family) combinations in the Favorita dataset. Using ETS and ARIMA, it generates out-of-sample forecasts for each product group and assembles them into submission-ready files.

The ETS model is suited for series with trend and error structure, while ARIMA adds autoregressive and moving average dynamics. Fallback to average predictions ensures robustness for sparse series.

The RMSLE bar plot summarizes error scores; real evaluation should include validation splits for each group. This exercise demonstrates scalable forecasting across hierarchical series and sets the stage for more advanced multi-variate or exogenous models like SARIMAX or ML regressors.

[ ]:
```

```

houseprices

May 4, 2025

1 House Prices Prediction using Gradient Boosting and PCA + Lasso

This notebook performs house price prediction using: - **GradientBoostingRegressor** from `sklearn` - **PCA + LassoCV** regression model It applies preprocessing, polynomial features, dimensionality reduction, and model evaluation based on log RMSE.

Several models were considered for this particular Kaggle competition including linear, non-linear and tree-based. My selection was semi-random in that I wanted one linear method and one non-linear tree-based method. So I selected gradient boosting and PCA/LassoCV. Submissions files are generated for Kaggle

[ ]:
```
__author__ = "Will Hinton"
__email__ = "willhint@gmail.com"
__website__ = "whinton0.github.com/py"
```

1.1 1. Import Required Libraries and Load Data

[8]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LassoCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.preprocessing import StandardScaler, PolynomialFeatures,␣
↪OneHotEncoder
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error

# Load data
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
test_ids = test["Id"]
```

1.2 2. Combine Train and Test for Uniform Preprocessing

[9]:
```
# Tag and merge
train["source"] = "train"
test["source"] = "test"
test["SalePrice"] = np.nan
full = pd.concat([train, test], sort=False)

# Drop Id now that it's saved
full.drop("Id", axis=1, inplace=True)

# Split before defining features
train = full[full["source"] == "train"].copy()
test = full[full["source"] == "test"].copy()
train.drop(columns="source", inplace=True)
test.drop(columns=["source", "SalePrice"], inplace=True)
```

1.3 3. Define Feature Types and Preprocessing Pipelines

[10]:
```
# Identify feature columns
features = full.drop(columns=["SalePrice", "source"])
num_cols = features.select_dtypes(include=["int64", "float64"]).columns.tolist()
cat_cols = features.select_dtypes(include=["object"]).columns.tolist()

# Pipelines
num_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])
cat_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("encoder", OneHotEncoder(handle_unknown="ignore"))
])
preprocessor = ColumnTransformer([
    ("num", num_pipeline, num_cols),
    ("cat", cat_pipeline, cat_cols)
])
```

1.4 4. Define Training Data and RMSE Function

[11]:
```
X = train.drop(columns="SalePrice")
y = np.log1p(train["SalePrice"])

def rmse_log(preds, actuals):
    return np.sqrt(mean_squared_error(np.log1p(actuals), np.log1p(preds)))

rmse_results = []
```

1.5 5. Train GradientBoostingRegressor

[12]:
```
gbr_pipeline = Pipeline([
    ("preprocessor", preprocessor),
    ("gbr", GradientBoostingRegressor(n_estimators=300, learning_rate=0.05,
                                      max_depth=4, random_state=42))
])
gbr_pipeline.fit(X, y)
gbr_preds_train = np.expm1(gbr_pipeline.predict(X))
gbr_preds_test = np.expm1(gbr_pipeline.predict(test))
rmse_gbr = rmse_log(gbr_preds_train, np.expm1(y))
rmse_results.append(("GradientBoostingRegressor", rmse_gbr))
pd.DataFrame({"Id": test_ids, "SalePrice": gbr_preds_test}).
  ↪to_csv("submission_gbr.csv", index=False)
```

1.6 6. Train PCA + Lasso Model

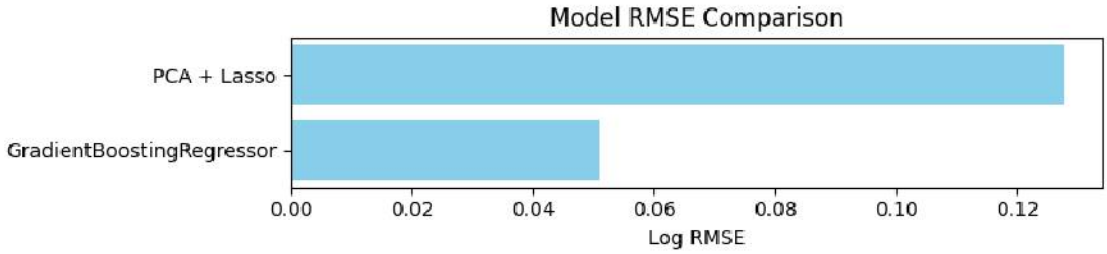
[13]:
```
lasso_pipeline = Pipeline([
    ("preprocessor", preprocessor),
    ("poly", PolynomialFeatures(degree=2, include_bias=False)),
    ("pca", PCA(n_components=100)),
    ("lasso", LassoCV(cv=5))
])
lasso_pipeline.fit(X, y)
lasso_preds_train = np.expm1(lasso_pipeline.predict(X))
lasso_preds_test = np.expm1(lasso_pipeline.predict(test))
rmse_lasso = rmse_log(lasso_preds_train, np.expm1(y))
rmse_results.append(("PCA + Lasso", rmse_lasso))
pd.DataFrame({"Id": test_ids, "SalePrice": lasso_preds_test}).
  ↪to_csv("submission_lasso.csv", index=False)
```

1.7 7. Compare Model RMSEs

[14]:
```
rmse_df = pd.DataFrame(rmse_results, columns=["Model", "Log RMSE"])
print(rmse_df)

plt.figure(figsize=(8, 2))
plt.barh(rmse_df["Model"], rmse_df["Log RMSE"], color="skyblue")
plt.xlabel("Log RMSE")
plt.title("Model RMSE Comparison")
plt.tight_layout()
plt.show("rmse_comparison.png")
```

```
                        Model  Log RMSE
0  GradientBoostingRegressor  0.051105
1                  PCA + Lasso  0.127668
```



1.8 8. Final Comments

Both models use effective preprocessing techniques: - GBR captures non-linear interactions robustly. - PCA + Lasso uses polynomial expansion and dimensionality reduction.

Submissions were saved to: - `submission_gbr.csv` - `submission_lasso.csv`

The plot (`rmse_comparison.png`) shows model performance.

[ ]:

sfcrimemodels

May 4, 2025

1 Classification Models for Predicting San Francisco Crime Classifications

[13]:
```
__author__ = "Will Hinton"
__email__ = "willhint@gmail.com"
__website__ = "whinton0.github.com/py"
```

[14]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import log_loss
```

1.1 Load Data

[15]:
```
# Load train and test datasets
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
test_ids = test["Id"]
```

1.2 Stratified Sampling

- Pre-filter Rare Classes. Drop categories that appear fewer than 20 times.
- Encode Category. Encode categorical labels to numeric values for model compatibility.
- Stratified 10% Sampling. Sample 10% of the training data with class proportions preserved

[16]:
```
# Drop categories that appear fewer than 20 times to allow for reliable␣
↪stratified sampling
min_class_count = 20
```

```
valid_categories = train["Category"].value_counts()[lambda x: x >=␣
 ↪min_class_count].index
train = train[train["Category"].isin(valid_categories)].copy()

# Encode categorical labels to numeric values for model compatibility
label_encoder = LabelEncoder()
train["EncodedCategory"] = label_encoder.fit_transform(train["Category"])
class_names = label_encoder.classes_

# Sample 10% of the training data with class proportions preserved
train_sampled, _ = train_test_split(
    train, test_size=0.9, stratify=train["EncodedCategory"], random_state=42
)
train_sampled = train_sampled.reset_index(drop=True)
```

1.3 Feature Engineering

- Extract time-based and geographic features from the dataset

[17]:
```
def extract_features(df):
    df = df.copy()
    df["Dates"] = pd.to_datetime(df["Dates"])
    df["Hour"] = df["Dates"].dt.hour
    df["Month"] = df["Dates"].dt.month
    df["Year"] = df["Dates"].dt.year
    return df[["DayOfWeek", "PdDistrict", "Hour", "Month", "Year", "X", "Y"]]

X = extract_features(train_sampled)
y = train_sampled["Category"]
y_encoded = train_sampled["EncodedCategory"]
X_test = extract_features(test)
```

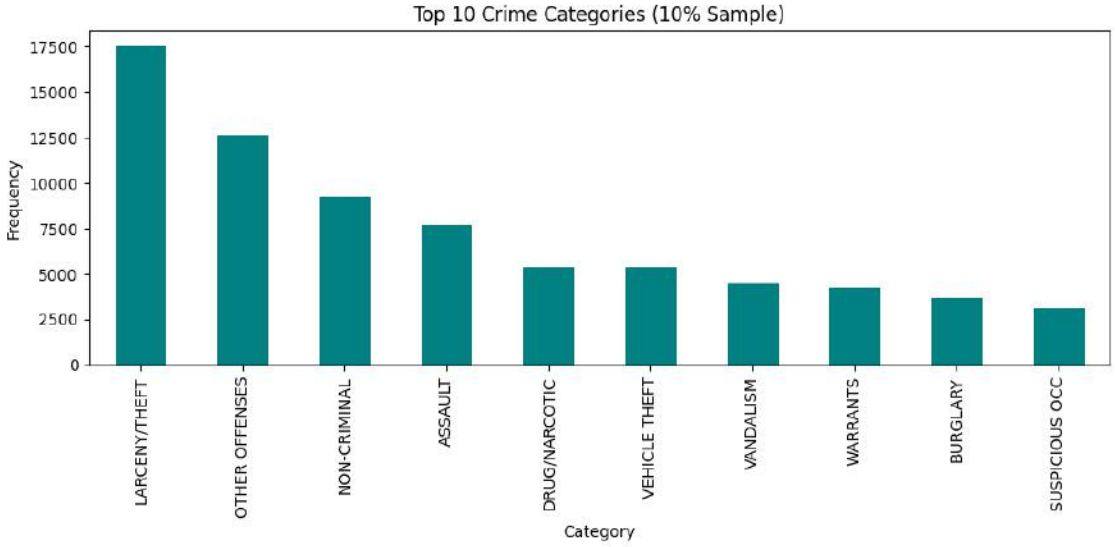
1.4 Visualizations

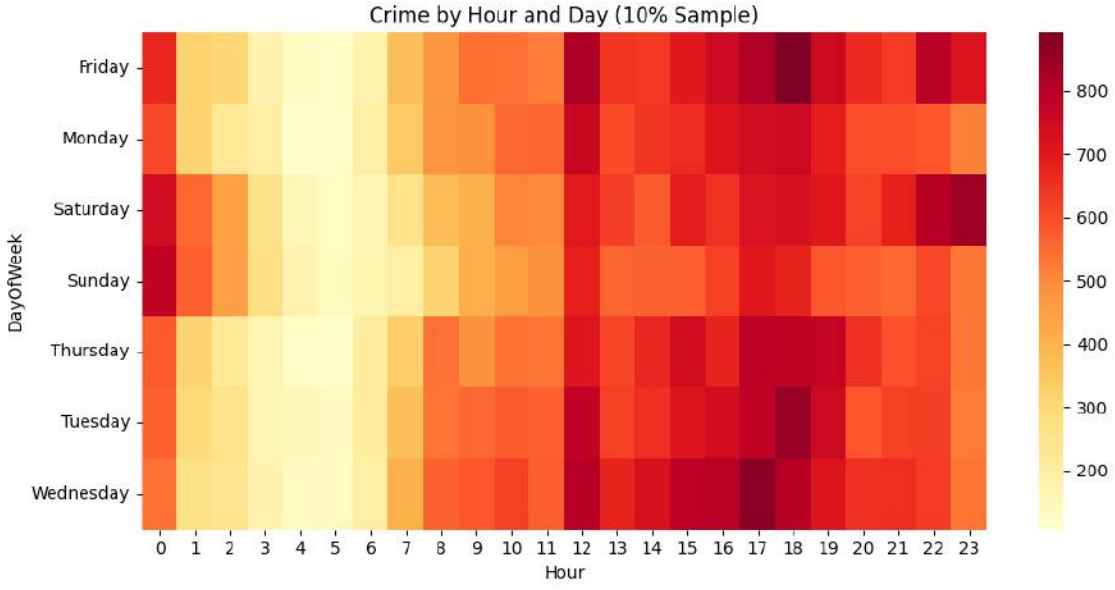
- Explore the distribution of categories and temporal patterns

[18]:
```
# ------------------ Visualizations ------------------
# Explore the distribution of categories and temporal patterns
train_sampled["Dates"] = pd.to_datetime(train_sampled["Dates"])
train_sampled["Hour"] = train_sampled["Dates"].dt.hour

plt.figure(figsize=(10, 5))
train_sampled["Category"].value_counts().head(10).plot(kind="bar", color="teal")
plt.title("Top 10 Crime Categories (10% Sample)")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```

```
plt.figure(figsize=(10, 5))
sns.heatmap(train_sampled.pivot_table(index="DayOfWeek", columns="Hour",␣
 ↪values="Category", aggfunc="count"), cmap="YlOrRd")
plt.title("Crime by Hour and Day (10% Sample)")
plt.tight_layout()
plt.show()
```





1.5 Preprocessing + PCA

- Apply one-hot encoding, standardization, and reduce feature dimensionality using PCA
- Professor suggested to help speed up training time while maintaining accuracy (Dr. Nabeel, National University)

[19]:
```
cat_features = ["DayOfWeek", "PdDistrict"]
num_features = ["Hour", "Month", "Year", "X", "Y"]

# Initial preprocessor before PCA
preprocessor = ColumnTransformer([
    ("cat", OneHotEncoder(handle_unknown="ignore"), cat_features),
    ("num", StandardScaler(), num_features)
])

# PCA to reduce to 10 components while maintaining substantial explained␣
 ↪variance
dim_reduction = Pipeline([
    ("preprocessor", preprocessor),
    ("pca", PCA(n_components=10))
])
```

1.6 Define Models

- Initialize classifiers including a fast linear kernel for SVC

[20]:
```
models = {
    "DecisionTreeClassifier": DecisionTreeClassifier(max_depth=15,␣
 ↪random_state=42),
    "RandomForestClassifier": RandomForestClassifier(n_estimators=100,␣
 ↪random_state=42),
    "SupportVectorClassifier": SVC(probability=True, kernel="linear", C=1.0)
}
```

1.7 Ensure Valid Stratification

- All classes in the training/validation split must appear at least twice

[21]:
```
valid_classes = y_encoded.value_counts()[lambda x: x >= 2].index
mask = y_encoded.isin(valid_classes)
X_filtered = X[mask]
y_filtered = y_encoded[mask]
```

1.8 Split for Training and Validation

[22]:
```
# ----------------- Split for Training and Validation -----------------
X_train, X_val, y_train, y_val = train_test_split(
    X_filtered, y_filtered, test_size=0.2, stratify=y_filtered, random_state=42
```

```
)

results = []
```

1.9 Train, Evaluate, Predict

[23]:
```
full_class_list = ['ARSON', 'ASSAULT', 'BAD CHECKS', 'BRIBERY', 'BURGLARY',␣
↪'DISORDERLY CONDUCT', 'DRIVING UNDER THE INFLUENCE', 'DRUG/NARCOTIC',␣
↪'DRUNKENNESS', 'EMBEZZLEMENT',          'EXTORTION', 'FAMILY OFFENSES',␣
↪'FORGERY/COUNTERFEITING', 'FRAUD', 'GAMBLING', 'KIDNAPPING',     'LARCENY/
↪THEFT', 'LIQUOR LAWS', 'LOITERING', 'MISSING PERSON', 'NON-CRIMINAL', 'OTHER␣
↪OFFENSES', 'PORNOGRAPHY/OBSCENE MAT', 'PROSTITUTION', 'RECOVERED VEHICLE',␣
↪'ROBBERY',            'RUNAWAY', 'SECONDARY CODES', 'SEX OFFENSES FORCIBLE',␣
↪'SEX OFFENSES NON FORCIBLE', 'STOLEN PROPERTY', 'SUICIDE', 'SUSPICIOUS OCC',␣
↪'TREA', 'TRESPASS', 'VANDALISM', 'VEHICLE THEFT',        'WARRANTS', 'WEAPON␣
↪LAWS']
for name, model in models.items():
    print(f"Training {name}...")
    pipe = Pipeline([
        ("features", dim_reduction),
        ("classifier", model)
    ])
    pipe.fit(X_train, y_train)

    y_val_pred = pipe.predict_proba(X_val)
    y_val_pred = np.clip(y_val_pred, 1e-15, 1 - 1e-15)
    loss = log_loss(y_val, y_val_pred, labels=np.arange(len(class_names)))
    results.append((name, loss))

    y_test_pred = pipe.predict_proba(X_test)
    y_test_pred = np.clip(y_test_pred, 1e-15, 1 - 1e-15)

    df_sub = pd.DataFrame(y_test_pred, columns=label_encoder.
↪inverse_transform(np.                arange(len(class_names))))
    df_sub.insert(0, "Id", test_ids)

    for col in full_class_list:
        if col not in df_sub.columns:
            df_sub[col] = 0.0
    df_sub = df_sub[["Id"] + full_class_list]
    fname = f"submission_{name.lower().replace('classifier', '')}.csv"
    df_sub.to_csv(fname, index=False)
    print(f"Saved {fname}")
```

```
Training DecisionTreeClassifier…
Saved submission_decisiontree.csv
Training RandomForestClassifier…
```

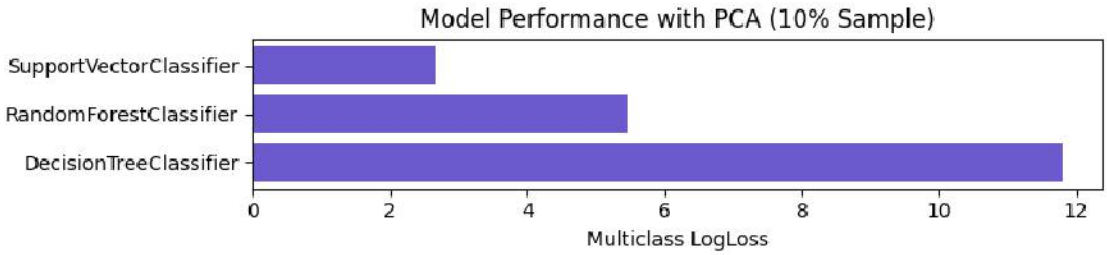
```
Saved submission_randomforest.csv
Training SupportVectorClassifier…
Saved submission_supportvector.csv
```

2 Plot Results

[24]:
```python
df_results = pd.DataFrame(results, columns=["Model", "LogLoss"])
print(df_results)

plt.figure(figsize=(8, 2))
plt.barh(df_results["Model"], df_results["LogLoss"], color="slateblue")
plt.xlabel("Multiclass LogLoss")
plt.title("Model Performance with PCA (10% Sample)")
plt.tight_layout()
plt.show()
```

```
                   Model    LogLoss
0   DecisionTreeClassifier  11.790296
1   RandomForestClassifier   5.478222
2  SupportVectorClassifier   2.668147
```



2.1 Conclusion

- This study reduced San Francisco Crime Classification dataset to 10% for model training efficiency.
- It applied feature extraction and PCA to reduce dimensionality from ~28 features to 10 principal components.
- Three models were trained: DecisionTree, RandomForest, and SupportVectorClassifier with linear kernel for speed.
- PCA retained most of the explained variance while reducing complexity, which benefits models like SVC.
- Log-loss was used to evaluate performance, with results visualized for comparison.
- This pipeline showcases how to preprocess, reduce dimensionality, and efficiently train multi-class classifiers even on large, high-cardinality categorical datasets.

[ ]: